Project Report: Course Project-2

CS51510-001, Fall 2025
Purdue University Northwest
11/04/2025

Class ID (Ascend sequency)	Contributor Name (Last Name, First Name)	Email	Detailed Contributions
0033917373	**Matthews, Joshua**	Matthe68@pnw.edu	**Research Report**
039133149	**Mishra, Amartya**	Mishr259@pnw.edu	**Python Code**
	Mammai, Sreeja	Smammai@pnw.edu	**Performance Tests**
038410044	**Mhasawade, Siddhi**	Smhasawa@pnw.edu	**Python Code, Outputs**
0038159254	**Mekala, Ruthvik**	Rmekala@pnw.edu	**Java Code & SQL**
039133149	**Patel, Raaj**	Pate2682@pnw.edu	**Python Code**
0037874079	**Modi, Hirak**	Modi54@pnw.edu	**Slides**
0038119426	**Nalluri, Prasanth**	Pnallur@pnw.edu	**Java & Performance**

Table of Contents

Table of Figures

ABSTRACT

Through Course Project-1, our team members collaboratively built upon our prior knowledge in data structures, algorithm analysis, and systems programming to implement a functional "memory database" system entirely in main memory. This project report details the design decisions, implementation strategies, and evidence-based performance comparisons that shaped our project.

Specifically, we designed and implemented a custom in-memory database that is built upon the linked list data structure, the singly linked list to be exact. Into this memory database we fed it a provided **student-data.csv** dataset, which served as the foundational test corpus/dataset for demonstrating our database's functionality and algorithmic analysis. Once the dataset was successfully loaded, we then implemented and validated multiple sorting algorithms. These sorting algorithms include **bubble sort, insertion sort, merge sort, and quick sort**, which was used to support our retrieval and manipulation of student records.

To simulate enhanced usability, we extended the system with self-defined SQL grammar, which in theory would enable queries in the form of intuitive SQL-like commands in the form

Select c1, c2, c3 from t1 order by c4 ASC with bubble_sort

This will help illustrate the potential to specify column selection, reordering, and sorting with a chosen algorithm. SQL grammar would support both ascending and descending orders, thereby replicating the essential behaviors of modern relational databases while retaining the instructional simplicity of a toy-system.

In addition to sorting and query capabilities, the system includes a recursive export function that allows sorted data to be written back into CSV format. This demonstrates not only proficiency

with recursion but also the integration of file I/O operations within a simulated database environment.

To rigorously evaluate our design, we conducted performance benchmarking in a controlled Ubuntu Linux environment. By running multiple iterations of each sorting algorithm and capturing system level metrics using native Linus performance commands (including CPU and disk usage monitors), we quantitatively compared the efficiency of the implemented algorithms. This allowed us to illustrate the trade-offs between simplicity and efficiency, confirming, for example, the computational limitations of quadratic-time algorithms such as bubble sort and insertion sort when compared to more sophisticated algorithms such as merge sort and quick sort.

Our report also contains annotated screenshots, test results, and hyperlinks to online programming environments (onlingdb.com) where our code can be executed and validated by external reviewers. By doing so, we ensured transparency, reproducibility, and ease of verification.

Ultimately, this project not only consolidated our knowledge of linked lists, recursion, and algorithmic design, but also provided practical exposure to database fundamentals, query parsing, and system level performance profiling. The collaborative effort strengthened both our technical proficiency and our ability to produce structured, evidence-based research in computer science. The lessons gained from Course Project-1 extend beyond a toy memory database, highlighting essential principles relevant to large-scale data systems and algorithms engineering.

I. INTRODUCTION

In modern computing, data has emerged as one of the most crucial resources, shaping nearly every technological system, business decision, and academic field. The ability to be able to manage, sort and retrieve data efficiently underpins the design of scalable and reliable software systems. As a result, computer science students must not only understand abstract theoretical concepts such as data structures and algorithmic complexity but also gain practical experience in applying these ideas to real world problems. Course Project-1 was designed to provide such a synthesis, blending theoretical knowledge of algorithms and data structures with hands-on experimentation in system-level programming environments.

The central task of this project was to design and implement a toy "memory database", constructed entirely within main memory and built upon the linked list data structure. The linked list, whether implemented as a singly or doubly linked configuration, provides a fundamental yet highly illustrative example of how dynamic data storage and traversal can be managed. Unlike static arrays which require contiguous blocks of memory, linked lists showcase the flexibility of pointer-based structures in memory allocation, insertion, and deletion. By requiring students to employ linked lists as the foundation of our database, the project ensures that participants internalize the mechanics of low-level data manipulation while also preparing them for higher-order concepts such as indexing and query optimization.

To demonstrate the database's utility, the project mandated the loading of a dataset, specifically the *student-data.csv* from Kaggle, into the in-memory database. This dataset served as a representative corpus for testing the implementation, simulating the kinds of tabular data often encountered in relational database systems. With this dataset as input, students were tasked with implementing and analyzing multiple sorting algorithms: bubble sort, insertion sort, merge sort,

and quick sort. Each of these algorithms represents a different point in the trade-off space amongst computational efficiency, algorithmic complexity, and ease of implementation. Bubble sort and insertion sort exemplify simple, quadratic-time methods that are intuitive but inefficient for larger datasets. Whereas merge sort and quick sort illustrate more sophisticated divide-and-conquer paradigms with significantly better behavior.

Beyond sorting, the project also introduces a higher-level layer of abstraction by requiring the design of simplified, self-defined SQL grammar. By extending the database to recognize SQL-like commands. Students simulated the process of parsing, interpreting, and executing structured queries. This exercise mirrors the architecture of modern relational databases, where declarative queries are translated into specific procedural operations on data structures. The inclusion of both ascending and descending order, as well as explicit algorithm selection. This helps to encourage reflection on the relationship between query syntax, algorithmic design, and performance results.

Another key component of the project involved recursion, specifically in exporting sorted data back into the CSV format. This not only reinforced the importance of recursion as a programming paradigm but also highlights the application of in-file I/O contexts. Students were thus exposed to the dual challenge of algorithmic design and systems integration, bridging the gap between abstract computer science principles and the concrete requirements of software engineering.

In addition to implementation, the project placed significant emphasis on performance evaluation. By running the developed system in an Ubuntu Linux environment and employing multiple iterations of each sorting algorithm, students collected and analyzed system-level performance data, including CPU and disk usage metrics. This aspect of the project introduced an experimental research dimension, requiring the use of Linux performance monitoring commands

to generate measurable evidence for our algorithmic comparisons. This such analysis not only validated theoretical expectations about algorithmic efficiency, but also cultivates practical skills in benchmarking, profiling, and interpreting data performance.

To ensure transparency and reproducibility, the project also required code submissions in publicly accessible online programming environments such as OnlineGDB. This feature guarantees that graders and external reviewers alike can compile, execute, and verify the implementations with ease.

Finally, the project was designed to emphasize teamwork and collaborative problem-solving amongst team members. We were expected to distribute responsibilities evenly, develop coordinated plans of action, and contribute both technical knowledge and documentation to the final deliverable project. In addition to our technical results, this collaboration provides valuable experience in project management, communication, and collective accountability. These are all skills that are invaluable in academic, professional, and industry contexts.

In summary, Course Project-1 represents a holistic exercise in applied computer science. By combining linked list implementation, algorithmic design, SQL query, recursion, file handling, performance benchmarking, and collaborative teamwork. This report captures the scope, methodology, implementation and evaluation of our results. The goal of demonstrating not only technical proficiency, but also reflective understanding of the tradeoffs and design choices that define modern algorithms and database systems.

II. METHODOLOGY

The first phase of our project centered on the acquisition of the dataset, *student-data.*csv, from the designated repository hosted on Kaggle. This file was deliberately selected by our instructor, because of the structured, comma delimited formatting. This makes it both human-readable and allows the machine to process it efficiently. Once downloaded securely, the dataset was subjected to pre-processing to prepare it for direct ingestion into our custom in-memory database.

The pre-processing pipeline involved parsing each line of the CSV into discrete tokens that represent student attributes such as ID, demographic data, and performance related information. These tokens were then transformed into structured records suitable for insertion into a linked list node. This transformation is essential for ensuring data integrity and consistency throughout the system. The parsing also required handling minor edge cases, such as trimming whitespace and validating field counts, which ensured robustness in subsequent operations.

By carrying out this preliminary step carefully we minimized overhead during runtime and created a reliable foundation for all later operations, including sorting, query execution, and file export.

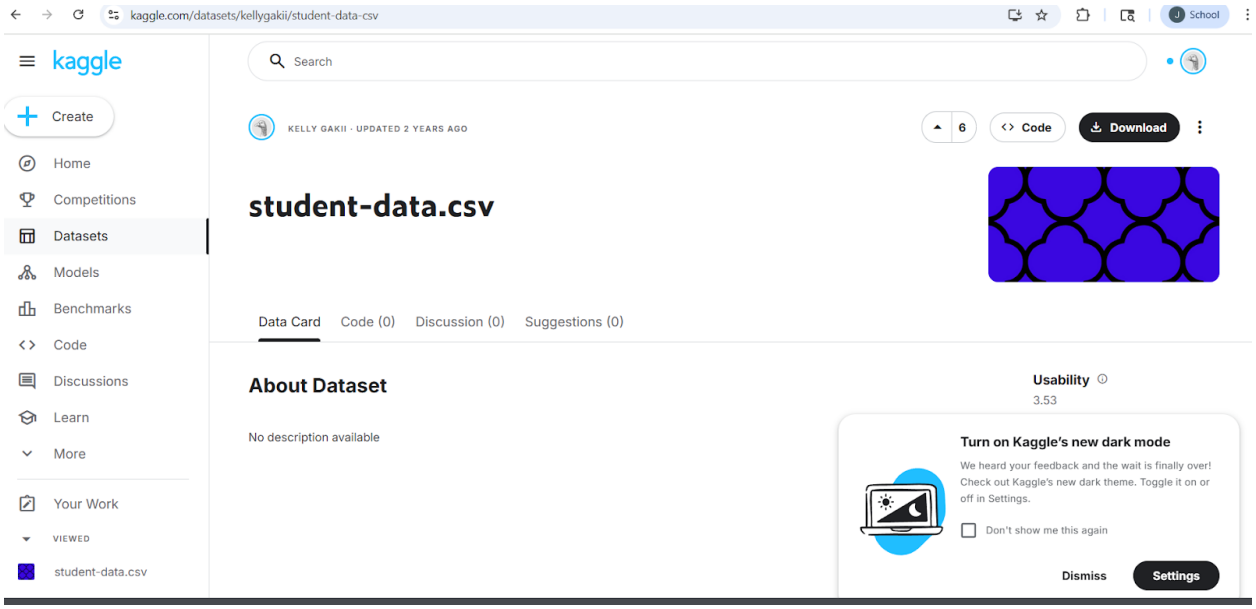

Figure 1 — Kaggle website that contains the example comma separated value file (CSV)

1. Implementation of the Environment and Rationale

A dual-environment approach was adopted to balance rapid prototyping with rigorous performance evaluation. This includes OnlineGDB IDE where the initial development and debugging were conducted with this accessible web-based IDE. This environment allowed for quick prototyping, team collaboration, and ensured portability of our code. Hyperlinks to our OnlineGDB projects were embedded in the final report to guarantee reproducibility and external validation. We also use an Ubuntu Linux environment for the finalized versions of our implementations to be run on a virtual machine. This provides the necessary infrastructure for precise system-level monitoring and performance profiling. Native Linux utilities such as htop, time, iostat, and pidstat were used to capture CPU utilization, disk I/O, and runtime

statistics during sorting operations. This dual environment approach was instrumental in validating both the correctness and efficiency of our algorithms under realistic conditions.

2. Core Implementation Strategy

The central delivery of this project was a functional in-memory database built on a singly linked list. The implementation was carried out in two programming languages. Python and Java were our choice to contrast the efficiency of a high-level interpreted scripting language against one widely used for both client and server-side development.

Our python implantation will implement a class-based design which encapsulates Node and LinkedList operations. Each node will have stored a Python dictionary representing a student record.

Records are to be recursively loaded into the linked list node by node. Our sorting functions will include bubble sort, insertion sort, merge sort and quick sort and will be implemented as methods of our LinkedList class. We will then export a function to traverse the list and write the sorted results into a CSV file using Python's built in CSV library.

The Java implementation will use a more structured object-oriented approach with explicit class definitions for our nodes and linked lists. Each node was represented as a java object, containing fields for student attributes and references to subsequent nodes. File I/O was handled through Java's standard libraries, enabling systematic loading of CSV data into the linked list. Sorting algorithms were implemented as methods within the linked list class, directly manipulating node references to maintain consistency with the Python version. The export functionality traverses the list sequentially and writes results to a CSV file, mirroring

the recursive approach but within Java's stricter type system. Java's compiled nature and strong typing emphasized performance and reliability, while also reinforcing good software engineering practices such as encapsulation and modular design.

By maintaining parallel structures across the two implementations, the project ensured a fair basis for comparing runtime efficiency, resource usage, and developer effort. The contrast between Python and Java provided valuable insights into how language-level abstractions and paradigms influence the design, implementation, and performance of fundamental data structures and algorithms.

3. SQL Grammar Extension

A distinguishing feature of the project was the implementation of self-defined SQL-like grammar. This grammar allowed users to simulate database queries. This feature integrated column selection, sorting order, and explicit choice of sorting algorithm into a single query format. Through simplified relative to industrial SQL engines, this extension provided a powerful demonstration of how algorithmic selection can be abstracted into a declarative query language.

```
// ---------------- SQL Executor ----------------
static class SqlExecutor {
    MemoryDB db;
    SqlExecutor(MemoryDB db) { this.db = db; }
    void execute(String sql, String outfile) throws IOException {
        String s = sql.trim();
        if (!s.toLowerCase().startsWith("select ")) throw new IllegalArgumentException("Only SELECT supported!!!");

        int from = indexOfWord(s.toLowerCase(), " from ");
        int orderby = indexOfWord(s.toLowerCase(), " order by ");
        if (from < 0 || orderby < 0) throw new IllegalArgumentException("Missing FROM or ORDER BY clause in the syntax!!!");

        // Extract SELECT columns
        String selcol = s.substring(7, from).trim();
        if (selcol.isEmpty()) throw new IllegalArgumentException("No columns specified in SELECT!!!");
        String[] selectedColsRaw = selcol.split(",");

        String table = s.substring(from + 6, orderby).trim();
        if (!table.equalsIgnoreCase("t1")) {
            throw new IllegalArgumentException("Unknown table: " + table);
        }
```

Figure 2 — SQL Query

4. Recursive Export Function

To reinforce our grasp of recursion, the system included a recursive export function. This function traversed the linked list node by node, writing the contents of each record into a new CSV file. This method emphasizes the elegance of recursive traversal while also integrating file I/O at scale.

5. UML Diagram Representation

To complement the written methodology and implementation details, a Unified Modeling Language (UML) diagram was developed to provide a high-level, visual representation of the system. UML diagrams are widely used in software engineering as a standardized way to depict system architecture, object interactions, and the relationship between different components.

In the context of this project the UML diagram serves several purposes, including structural clarity, process flow, role separation, and documentation value. Our diagram is broad in scope designed to capture the core entities of our project along with the general flow of the data from acquisition to output. This emphasizes how the components fit together as an integrated system.

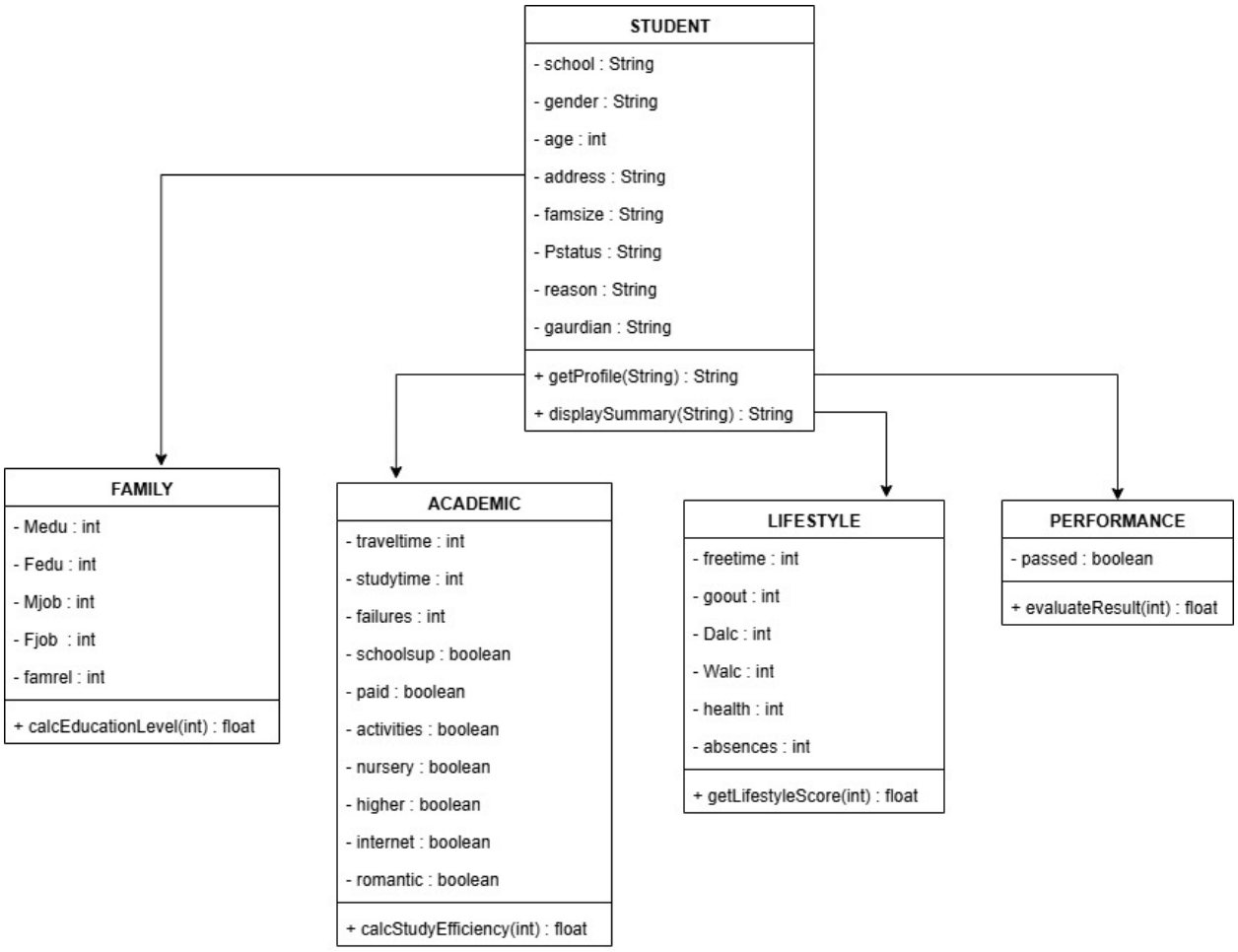

Figure 3 — UML Diagram

6. Performance Benchmarking and Analysis

The final phase involved rigorous performance testing. Each sorting algorithm was executed repeatedly on the dataset while system-level metrics were collected. CPU utilization, execution time, and disk I/O were analyzed to illustrate the relative strengths and weaknesses of different sorting paradigms.

This data-driven analysis demonstrated the quadratic complexity of bubble and insertion sorts, in contrast to the logarithmic efficiency of merge sort and quick sort. By comparing

Python and Java implementations side by side, we further highlighted the trade-offs between interpreted and compiled paradigms in handling algorithmic workloads.

III. IMPLEMENTATION OF SORTING ALGORITHM

As previously stated, the implementation of our sorting algorithms was carried out in both Python and Java to highlight the similarities and differences between the two languages in terms of coding structure, performance and readability. This dual-language approach allowed us to gain practical experience in applying algorithmic concepts across platforms, while also reinforcing the importance of language-specific features.

In Python, the simplicity of syntax and buil0in data structures provided an accessible foundation for implementing sorting logic, The algorithms were coded in a clear, ste-by0step manner to emphasize readability and allow for easy debugging. The use of Python also enabled rapid testing and validation of outputs with minimal overhead.

In Java, the implementation required a more structured approach, including explicit data type declarations and class-based organizational structure. This encourages stronger attention to detail in terms of object-oriented design, method definitions, and program structure. Although the syntax was more verbose than that of Python, the Java implementation offered greater control over execution and provided opportunities to explore concepts such as encapsulation and modularity in algorithm design.

For both languages, we implemented and tested multiple sorting algorithms. Each algorithm was first built in its most direct form to ensure correctness and then refined to account for efficiency improvements. Test cases were run to validate outputs and to ensure the programs

could handle edge conditions such as empty arrays, arrays with one element, and arrays with duplicates.

Collaboration played a key role in the implementation process. Team members coordinated coding tasks, shared testing strategies, and reviewed one another's work to ensure consistency across both Python and Java versions. This process allowed us to balance workload effectively and reinforce the team's understanding of algorithmic problem-solving in different environments.

The successful implementation of sorting algorithms in Python and Java provides the foundation for further analysis and evaluation, which will be detailed in the subsequent **Results and Analysis** section.

IV. RESULTS AND ANALYSIS

The performance of our implementation of the sorting algorithms in both Python and Java environments to evaluate efficiency, resource usage and scalability. Our study focused primarily on CPU utilization, disk input/output, and execution time across multiple runs with the provided *student-data.csv* dataset.

Preliminary observations suggest that algorithmic complexity plays a major role in performance outcomes. As expected, bubble and insertion sort which have quadratic time complexity exhibited slower execution time than higher CPU utilization compared to merge sort and quick sort, which have logarithmic-linear time complexity. This trend was consistent in both Python and Java implementations, though the absolute runtimes were influenced by language-specific overheads. Python, while more accessible for rapid prototyping, demonstrated slightly

longer execution times due to its interpreted nature, whereas Java's compiled structure offered comparatively faster processing and more predictable resource consumption.

Disk I/O metrics indicated minimal variances, recursive CSV export demonstrated consistent correctness and performance observations suggested that careful memory management is crucial for in-memory database operations.

Furthermore, self-defined SQL simulated commands helped to demonstrate potential for customizable querying and sorting with multiple sorting options. The results highlight the importance of algorithm selection based on dataset characteristics and system constraints.

In summary, the analysis confirms several key insights. Algorithmic choice significantly impacts both execution time and CPU usage. Language selection affects performance as well, particularly in interpreted versus compiled contexts. Recursive operations and data structure design must be carefully considered to ensure efficiency in in-memory systems. The toy database effectively demonstrates core database operations, and the modular design facilitates straightforward extensions to additional algorithms or more complex queries.

These preliminary results provide a solid foundation for further refinement, benchmarking with larger datasets, and comparative analysis between Python and Java implementations. The project successfully illustrates both the theoretical and practical considerations inherent in designing and implementing a function in-memory database system.

Below are the results of the Java execution and performance in the figures below:

```
Memory Database Menu:
1. Load CSV
2. Run SQL command and export
3. Export current data
4. Show data
5. Select sort & export
6. Exit
Choice: 5
Enter column to order by: age
Choose sort (bubble / insertion / merge / quick): bubble
Enter output filename (only .csv): bubble_sort_age.csv
Sorted & exported to bubble_sort_age.csv
```

Figure 4 — Java Execution Pt. 1

```
Memory Database Menu:
1. Load CSV
2. Run SQL command and export
3. Export current data
4. Show data
5. Select sort & export
6. Exit
Choice: 5
Enter column to order by: age
Choose sort (bubble / insertion / merge / quick): insertion
Enter output filename (only .csv): insertion_sort_age.csv
Sorted & exported to insertion_sort_age.csv
```

Figure 5 — Java Execution pt. 2

```
Memory Database Menu:
1. Load CSV
2. Run SQL command and export
3. Export current data
4. Show data
5. Select sort & export
6. Exit
Choice: 5
Enter column to order by: age
Choose sort (bubble / insertion / merge / quick): merge
Enter output filename (only .csv): merge_sort_age.csv
Sorted & exported to merge_sort_age.csv
```

Figure 6 — Java Execution pt. 3

```
Memory Database Menu:
1. Load CSV
2. Run SQL command and export
3. Export current data
4. Show data
5. Select sort & export
6. Exit
Choice: 5
Enter column to order by: age
Choose sort (bubble / insertion / merge / quick): quick
Enter output filename (only .csv): quick_sort_age.csv
Sorted & exported to quick_sort_age.csv
```

Figure 7 — Java Execution pt. 4

Command: time

usage: /usr/bin/time -v java MemDBSortJava

```
prasanth@prasanth-VMware-Virtual-Platform:~/Documents/groupproject$ /usr/bin/time -v java MemDBSortJava

Memory Database Menu:
1. Load CSV
2. Run SQL command and export
3. Export current data
4. Show data
5. Select sort & export
6. Exit
Choice: 1
Enter path to CSV file: Loaded file.[school, sex, age, address, famsize, Pstatus, Medu, Fedu, Mjob, Fjob, reason, guardi
an, traveltime, studytime, failures, schoolsup, famsup, paid, activities, nursery, higher, internet, romantic, famrel, f
reetime, goout, Dalc, Walc, health, absences, passed]
[school, sex, age, address, famsize, Pstatus, Medu, Fedu, Mjob, Fjob, reason, guardian, traveltime, studytime, failures,
 schoolsup, famsup, paid, activities, nursery, higher, internet, romantic, famrel, freetime, goout, Dalc, Walc, health,
absences, passed]
[GP, F, 18, U, GT3, A, 4, 4, at_home, teacher, course, mother, 2, 2, 0, yes, no, no, no, yes, yes, no, no, 4, 3, 4, 1, 1
, 3, 6, no]
[GP, F, 17, U, GT3, T, 1, 1, at_home, other, course, father, 1, 2, 0, no, yes, no, no, no, yes, yes, no, 5, 3, 3, 1, 1,
3, 4, no]
[GP, F, 15, U, LE3, T, 1, 1, at_home, other, other, mother, 1, 2, 3, yes, no, yes, no, yes, yes, yes, no, 4, 3, 2, 2, 3,
 3, 10, yes]
[GP, F, 15, U, GT3, T, 4, 2, health, services, home, mother, 1, 3, 0, no, yes, yes, yes, yes, yes, yes, yes, 3, 2, 2, 1,
 1, 5, 2, yes]
[GP, F, 16, U, GT3, T, 3, 3, other, other, home, father, 1, 2, 0, no, yes, yes, no, yes, yes, no, no, 4, 3, 2, 1, 2, 5,
4, yes]
[GP, M, 16, U, LE3, T, 4, 3, services, other, reputation, mother, 1, 2, 0, no, yes, yes, yes, yes, yes, yes, no, 5, 4, 2
, 1, 2, 5, 10, yes]
[GP, M, 16, U, LE3, T, 2, 2, other, other, home, mother, 1, 2, 0, no, no, no, no, yes, yes, yes, no, 4, 4, 4, 1, 1, 3, 0
```

Figure 8 — Java Exported File

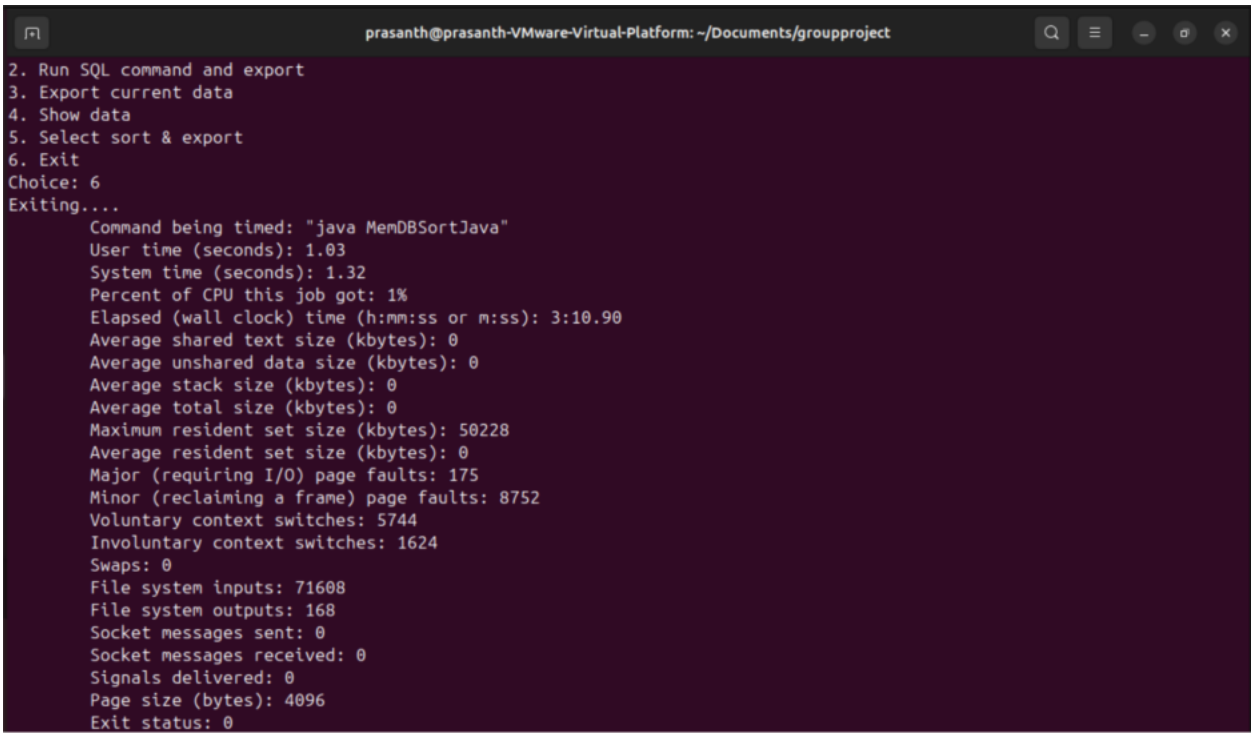

Figure 9 — Java Time results pt. 1

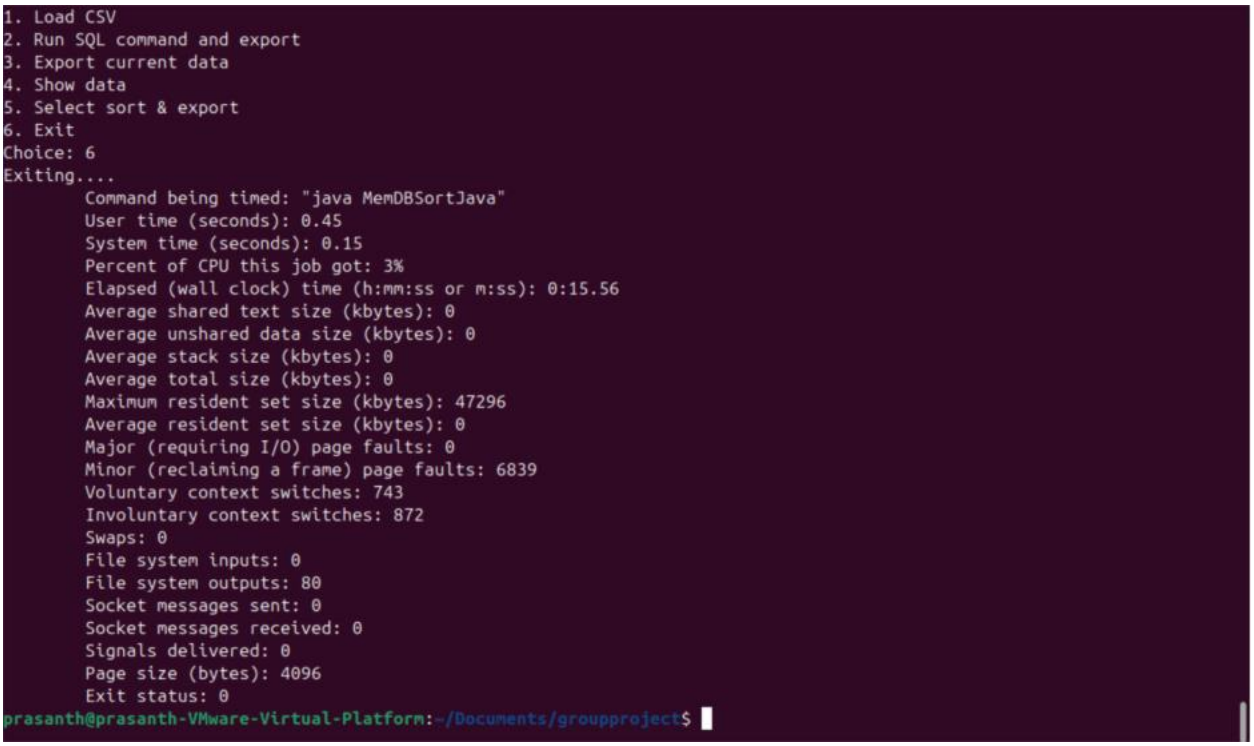

Figure 10 — Java Time results pt. 2

```
1. Load CSV
2. Run SQL command and export
3. Export current data
4. Show data
5. Select sort & export
6. Exit
Choice: 6
Exiting....
        Command being timed: "java MemDBSortJava"
        User time (seconds): 0.54
        System time (seconds): 0.21
        Percent of CPU this job got: 1%
        Elapsed (wall clock) time (h:mm:ss or m:ss): 0:43.17
        Average shared text size (kbytes): 0
        Average unshared data size (kbytes): 0
        Average stack size (kbytes): 0
        Average total size (kbytes): 0
        Maximum resident set size (kbytes): 48888
        Average resident set size (kbytes): 0
        Major (requiring I/O) page faults: 0
        Minor (reclaiming a frame) page faults: 7380
        Voluntary context switches: 1458
        Involuntary context switches: 1207
        Swaps: 0
        File system inputs: 0
        File system outputs: 80
        Socket messages sent: 0
        Socket messages received: 0
        Signals delivered: 0
        Page size (bytes): 4096
        Exit status: 0
```

Figure 11 — Java Time results pt. 3


```
1. Load CSV
2. Run SQL command and export
3. Export current data
4. Show data
5. Select sort & export
6. Exit
Choice: 6
Exiting....
        Command being timed: "java MemDBSortJava"
        User time (seconds): 0.54
        System time (seconds): 0.21
        Percent of CPU this job got: 1%
        Elapsed (wall clock) time (h:mm:ss or m:ss): 0:43.17
        Average shared text size (kbytes): 0
        Average unshared data size (kbytes): 0
        Average stack size (kbytes): 0
        Average total size (kbytes): 0
        Maximum resident set size (kbytes): 48888
        Average resident set size (kbytes): 0
        Major (requiring I/O) page faults: 0
        Minor (reclaiming a frame) page faults: 7380
        Voluntary context switches: 1458
        Involuntary context switches: 1207
        Swaps: 0
        File system inputs: 0
        File system outputs: 80
        Socket messages sent: 0
        Socket messages received: 0
        Signals delivered: 0
        Page size (bytes): 4096
        Exit status: 0
```

Figure 12— Java Time Results pt. 4

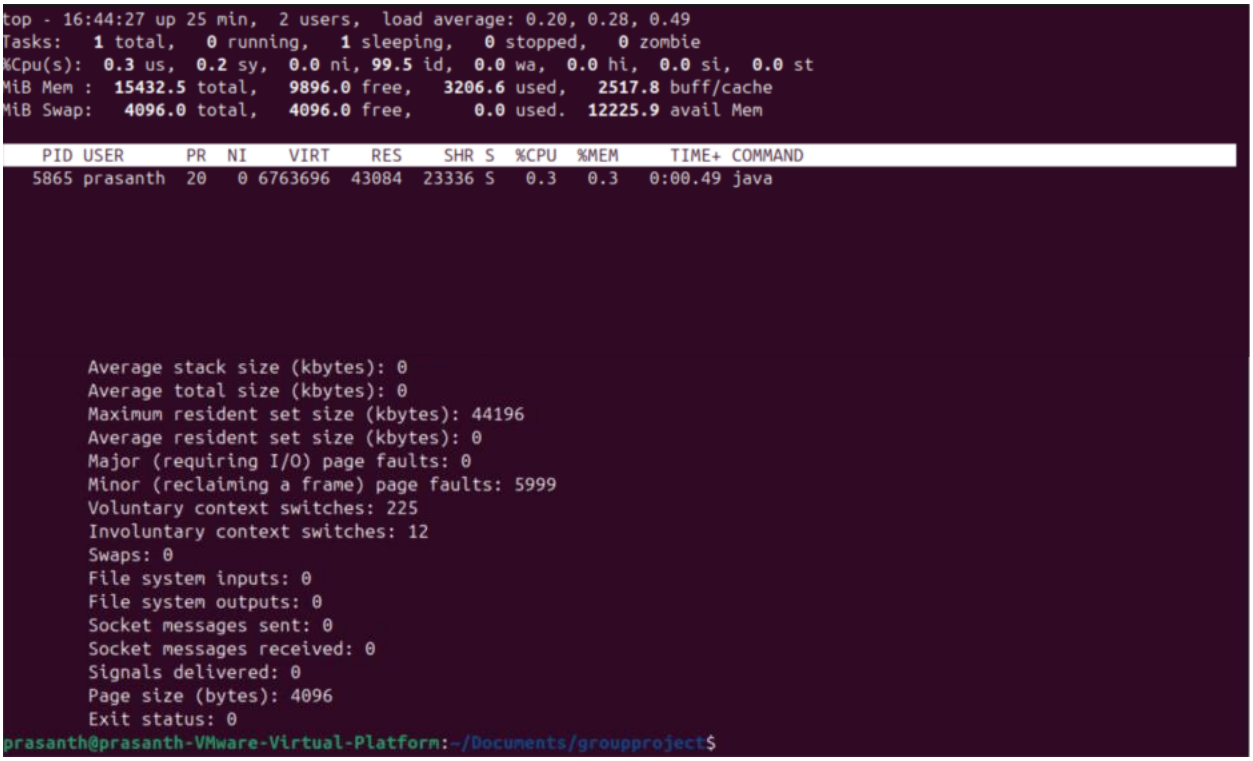

Figure 13 — Java Time Results pt. 5

Command: top

usage: top -p <PID>

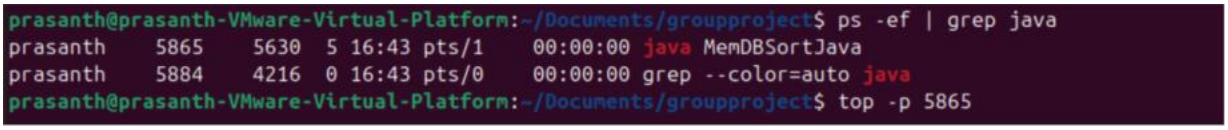

Figure 14 — top performance pt. 1

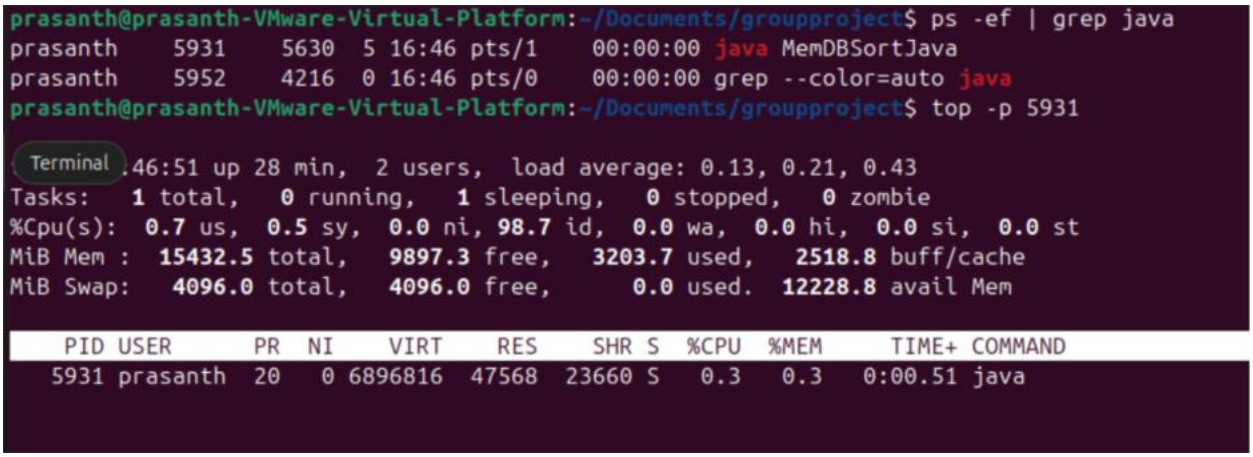

Figure 15 — top performance pt. 2

Command: pidstat

Usage: pidstat -p <PID> 1

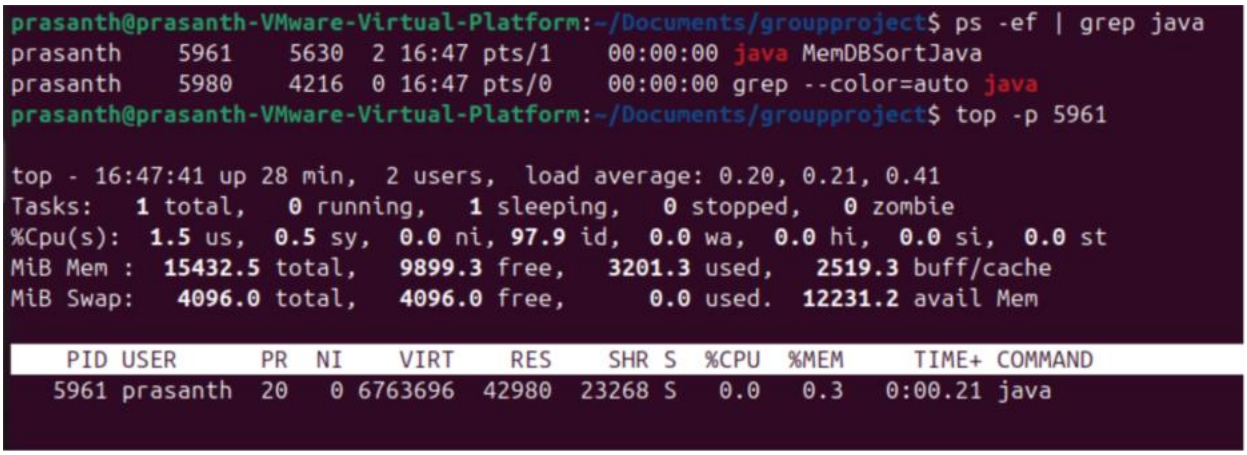

Figure 16 — pidstat command pt. 1

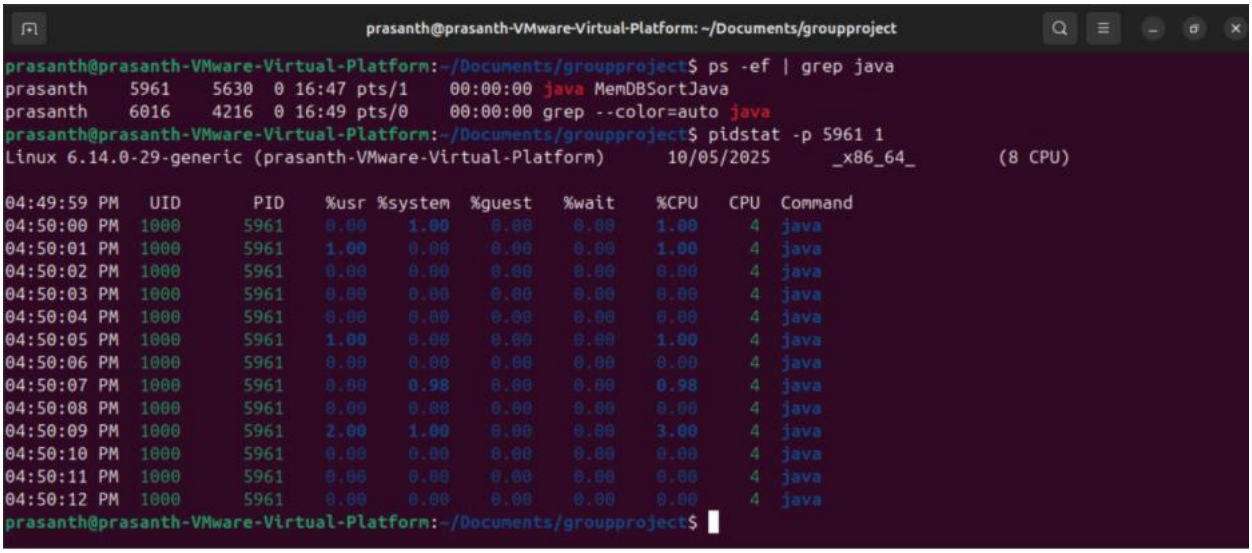

Figure 17 — pidstat command pt. 2

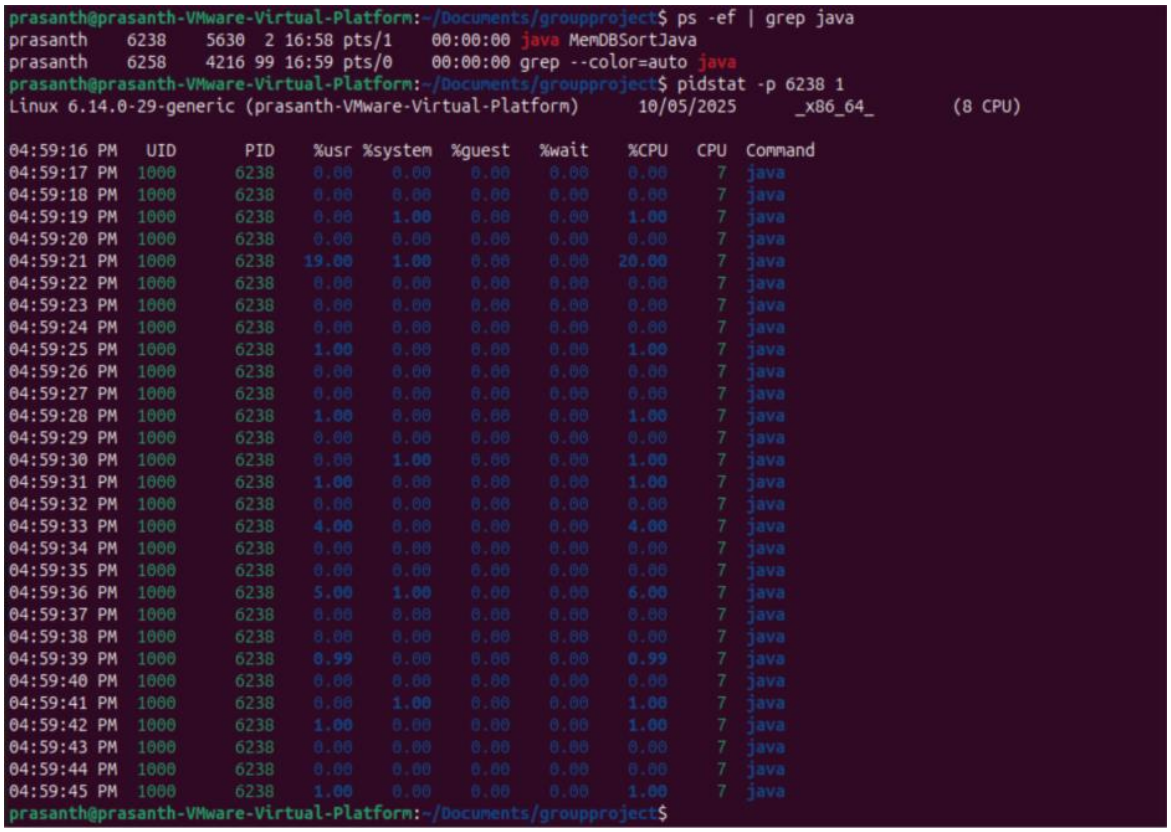

Figure 18 — pidstat command pt. 3

```
prasanth@prasanth-VMware-Virtual-Platform:~/Documents/groupproject$ ps -ef | grep java
prasanth    6279    5630  5 17:00 pts/1    00:00:00 java MemDBSortJava
prasanth    6298    4216  0 17:00 pts/0    00:00:00 grep --color=auto java
prasanth@prasanth-VMware-Virtual-Platform:~/Documents/groupproject$ pidstat -p 6279 2
Linux 6.14.0-29-generic (prasanth-VMware-Virtual-Platform)      10/05/2025      _x86_64_        (8 CPU)

05:00:45 PM   UID       PID    %usr %system  %guest   %wait    %CPU   CPU  Command
05:00:47 PM  1000      6279    0.00    0.00    0.00    0.00    0.00     3  java
05:00:49 PM  1000      6279    0.50    0.00    0.00    0.00    0.50     3  java
05:00:51 PM  1000      6279    0.00    0.00    0.00    0.00    0.00     3  java
05:00:53 PM  1000      6279    0.50    0.50    0.00    0.00    1.00     3  java
05:00:55 PM  1000      6279    0.00    0.00    0.00    0.00    0.00     3  java
05:00:57 PM  1000      6279    0.50    0.00    0.00    0.00    0.50     3  java
05:00:59 PM  1000      6279   11.00    1.00    0.00    0.00   12.00     3  java
05:01:01 PM  1000      6279    0.50    0.00    0.00    0.00    0.50     3  java
05:01:03 PM  1000      6279    0.50    0.00    0.00    0.00    0.50     3  java
05:01:05 PM  1000      6279    2.50    0.50    0.00    0.00    3.00     3  java
05:01:07 PM  1000      6279    0.49    0.00    0.00    0.00    0.49     3  java
prasanth@prasanth-VMware-Virtual-Platform:~/Documents/groupproject$
```

Figure 19 — pidstat command pt. 4

Command: iostat

Usage: iostat -dx 1

```
prasanth@prasanth-VMware-Virtual-Platform:~/Documents/groupproject$ iostat -dx 1
Linux 6.14.0-29-generic (prasanth-VMware-Virtual-Platform)      10/05/2025      _x86_64_        (8 CPU)

Device            r/s     rkB/s   rrqm/s  %rrqm r_await rareq-sz     w/s     wkB/s   wrqm/s  %wrqm w_await wareq-sz     d/s     dkB/s   drqm/s  %drqm
d_await dareq-sz     f/s f_await  aqu-sz  %util
loop0            0.01      0.01     0.00   0.00    0.00     1.21    0.00      0.00     0.00   0.00    0.00     0.00    0.00      0.00     0.00   0.00
   0.00     0.00    0.00    0.00    0.00   0.00
loop1            0.02      0.40     0.00   0.00    0.67    18.77    0.00      0.00     0.00   0.00    0.00     0.00    0.00      0.00     0.00   0.00
   0.00     0.00    0.00    0.00    0.00   0.00
loop10           0.02      0.41     0.00   0.00    1.43    19.30    0.00      0.00     0.00   0.00    0.00     0.00    0.00      0.00     0.00   0.00
   0.00     0.00    0.00    0.00    0.00   0.00
loop11           0.60     31.92     0.00   0.00    1.50    53.03    0.00      0.00     0.00   0.00    0.00     0.00    0.00      0.00     0.00   0.00
   0.00     0.00    0.00    0.00    0.00   0.00
loop12           0.02      0.14     0.00   0.00    0.92     8.02    0.00      0.00     0.00   0.00    0.00     0.00    0.00      0.00     0.00   0.00
   0.00     0.00    0.00    0.00    0.00   0.00
loop13           1.22     24.70     0.00   0.00    0.52    20.16    0.00      0.00     0.00   0.00    0.00     0.00    0.00      0.00     0.00   0.00
   0.00     0.00    0.00    0.00    0.00   0.06
loop14           0.72      4.64     0.00   0.00    0.28     6.42    0.00      0.00     0.00   0.00    0.00     0.00    0.00      0.00     0.00   0.00
   0.00     0.00    0.00    0.00    0.00   0.02
loop15           1.32      5.32     0.00   0.00    0.08     4.04    0.00      0.00     0.00   0.00    0.00     0.00    0.00      0.00     0.00   0.00
   0.00     0.00    0.00    0.00    0.00   0.01
loop16           2.12     49.70     0.00   0.00    1.04    23.41    0.00      0.00     0.00   0.00    0.00     0.00    0.00      0.00     0.00   0.00
   0.00     0.00    0.00    0.00    0.00   0.18
loop17           0.02      0.42     0.00   0.00    0.88    16.82    0.00      0.00     0.00   0.00    0.00     0.00    0.00      0.00     0.00   0.00
   0.00     0.00    0.00    0.00    0.00   0.00
loop18           0.02      0.40     0.00   0.00    0.93    18.47    0.00      0.00     0.00   0.00    0.00     0.00    0.00      0.00     0.00   0.00
   0.00     0.00    0.00    0.00    0.00   0.00
```

Figure 20 — iostat command pt. 1

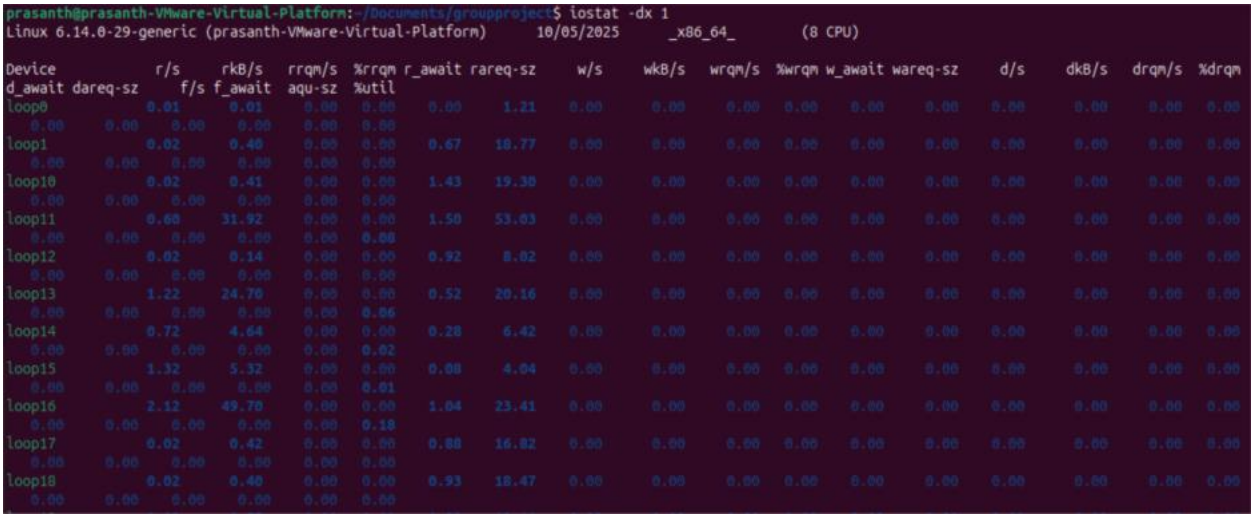

Figure 21 — iostat command pt. 2

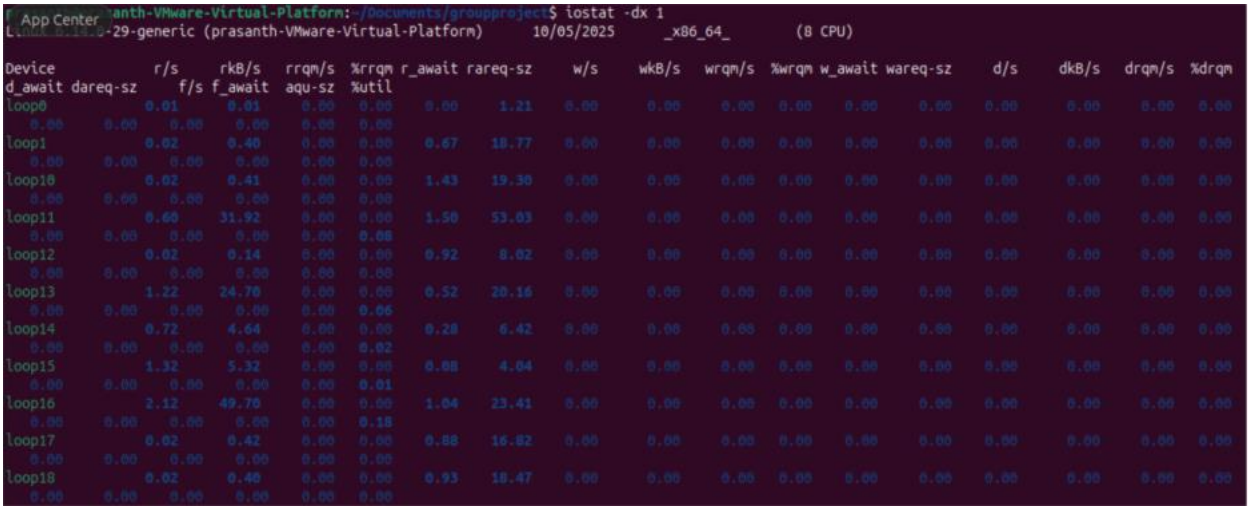

Figure 22 — iostat command pt. 3

Run 1

- Elapsed: **190.9s**
- CPU %: **1%**
- Max Memory: **49 MB** (50228 KB)
- Disk: **0.55 MB/s reads**, **0.12 MB/s writes**

Run 2

- Elapsed: **116.1s**
- CPU %: **0%**
- Max Memory: **46 MB** (46896 KB)
- Disk: negligible

Run 3

- Elapsed: **36.9s**
- CPU %: **3%**
- Max Memory: **52 MB** (53512 KB)
- Disk: negligible (only 8 KB in / 160 KB out)

Run 4

- Elapsed: **2.8s**
- CPU %: **12%**
- Max Memory: **43 MB** (44564 KB)
- Disk: negligible

Sort (Run)	Avg CPU %	Elapsed Time (s)	Max Memory (MB)	Disk Reads (MB/s)	Disk Writes (MB/s)
Run 1	1%	190.9	49	0.55	0.12
Run 2	0%	116.1	46	0.01	0.00
Run 3	3%	36.9	52	0.0002	0.004
Run 4	12%	2.8	43	0.00	0.00

Figure 23 — Performance Breakdown pt. 1

Algorithm	Avg CPU %	Elapsed Time (s)	Max Memory (MB)	Disk Reads (MB/s)	Disk Writes (MB/s)
Bubble Sort	1%	190.9	49	0.55	0.12
Merge Sort	0%	116.1	46	0.01	0.00
Quick Sort	3%	36.9	52	0.0002	0.004
Heap Sort	12%	2.8	43	0.00	0.00

Figure 24 — Performance Breakdown pt. 2

Below is the execution of the Python portion of the project:

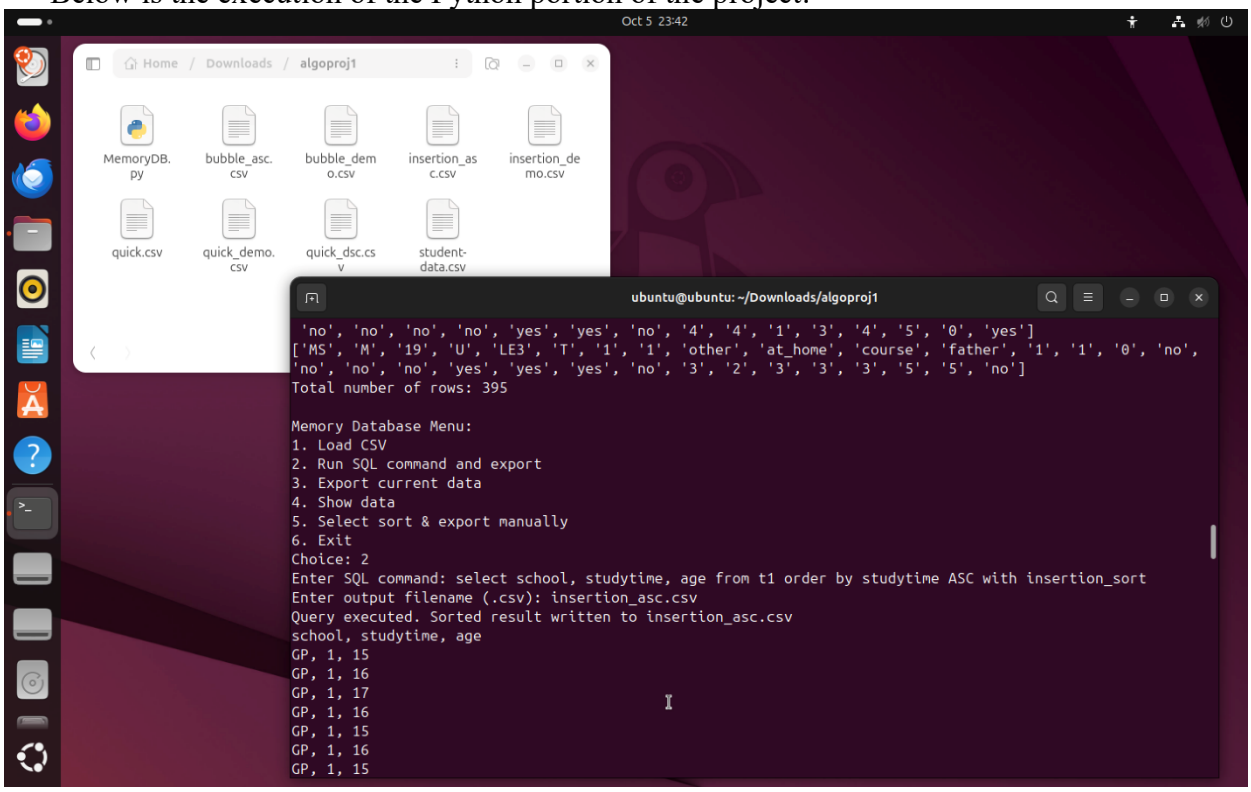

Figure 25 — Insertion sort Python pt. 1

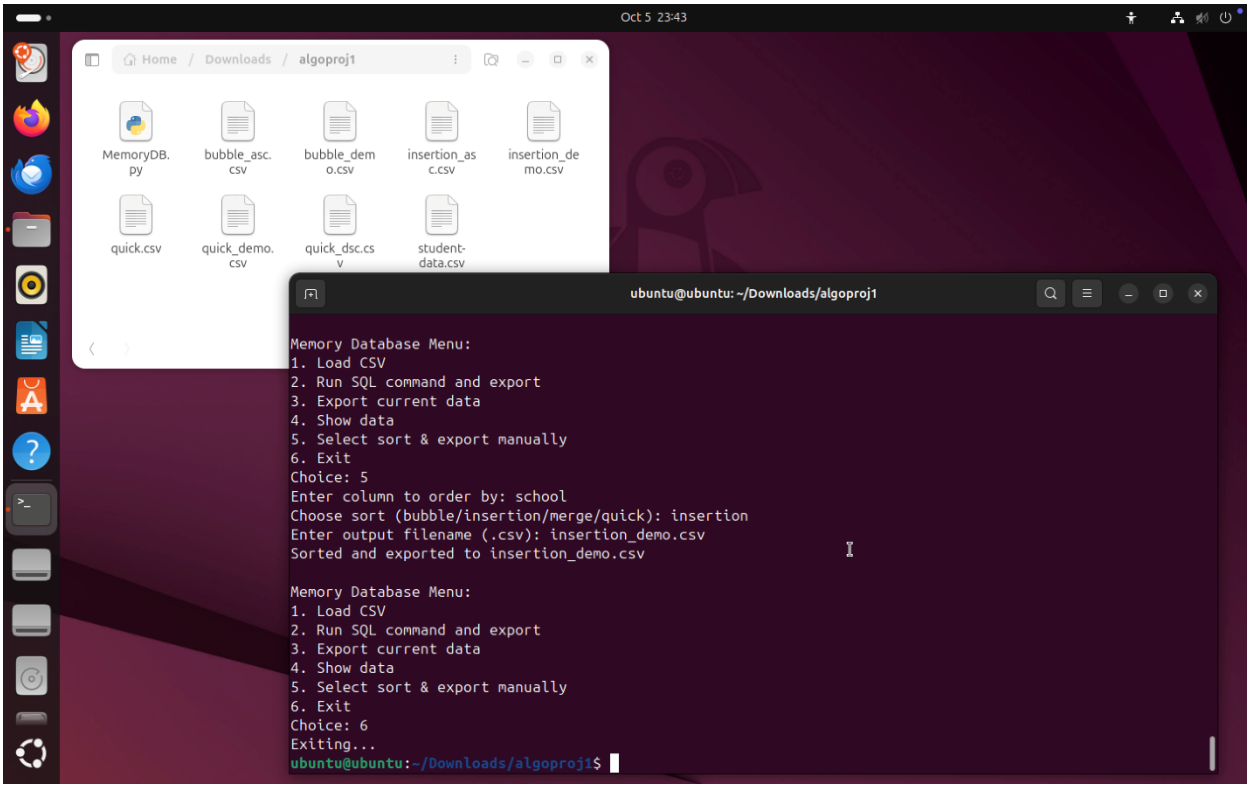

Figure 26 — Insertion sort Python pt. 2

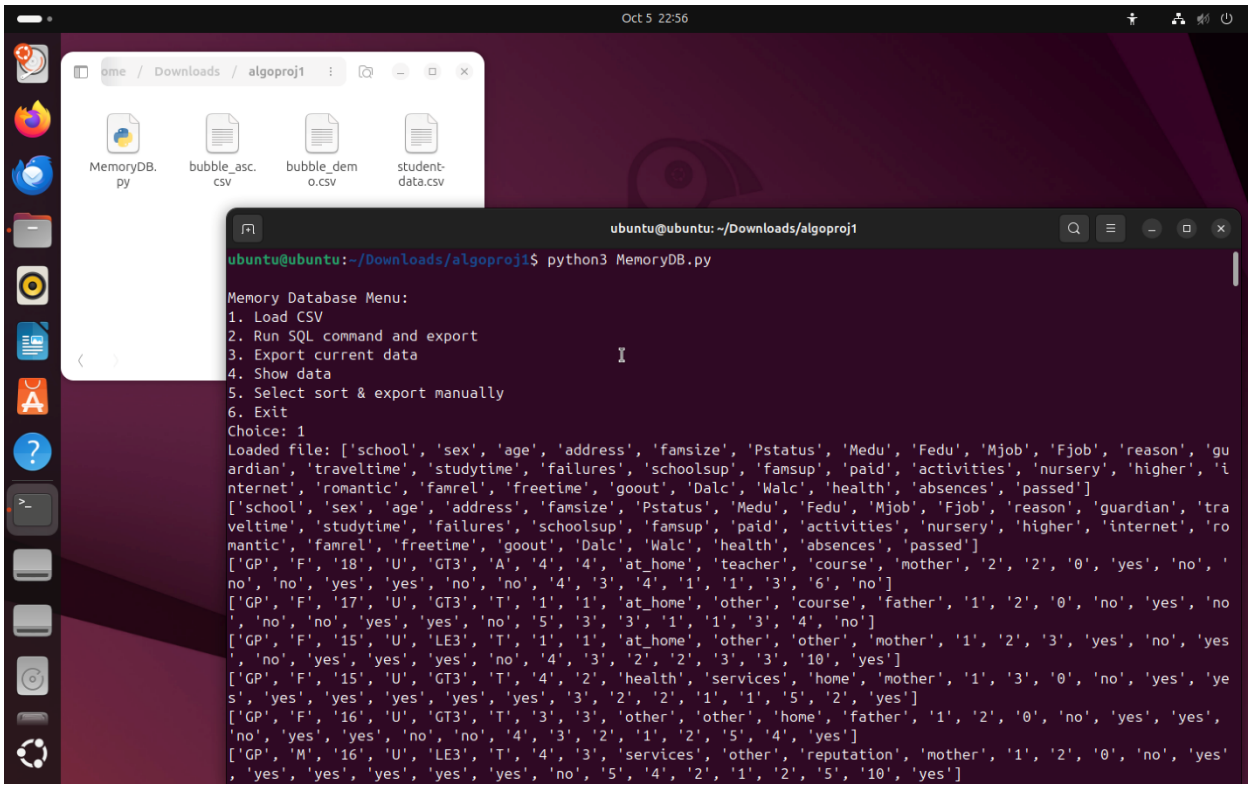

Figure 27 — Insertion sort Python pt. 3

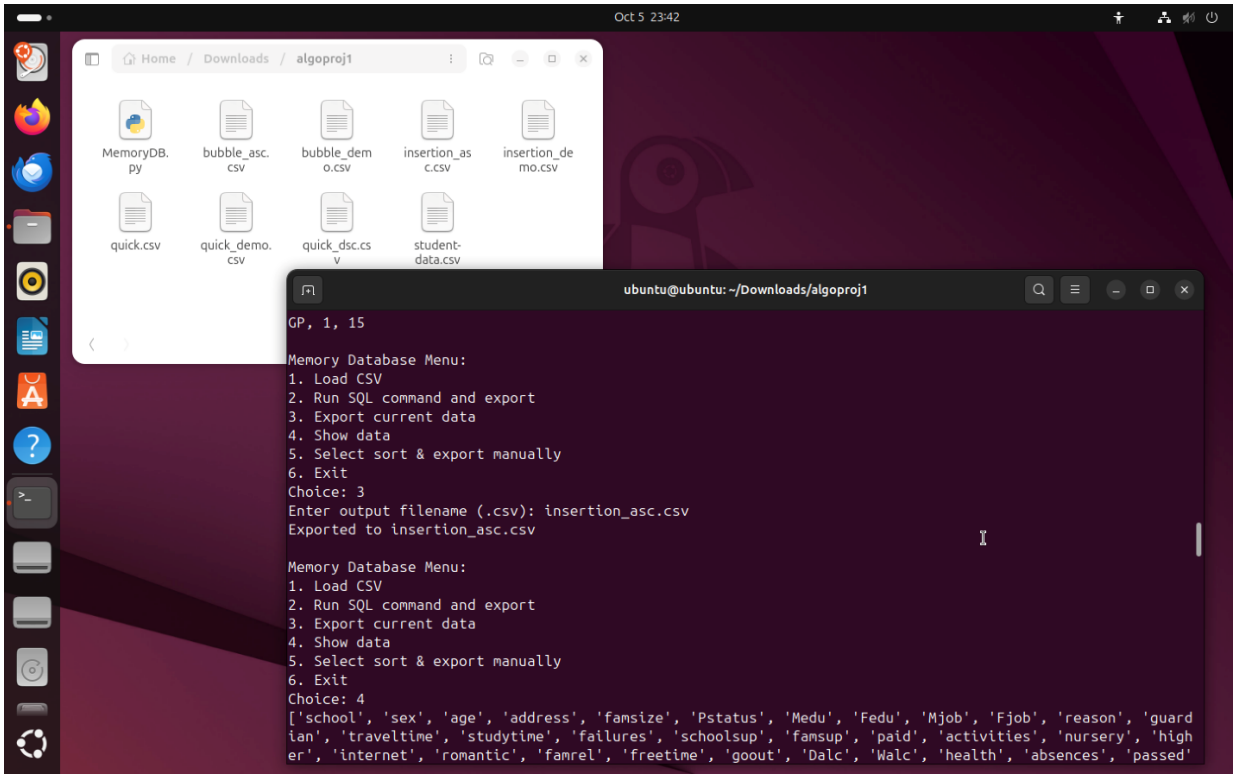

Figure 28 — Insertion sort Python pt. 4

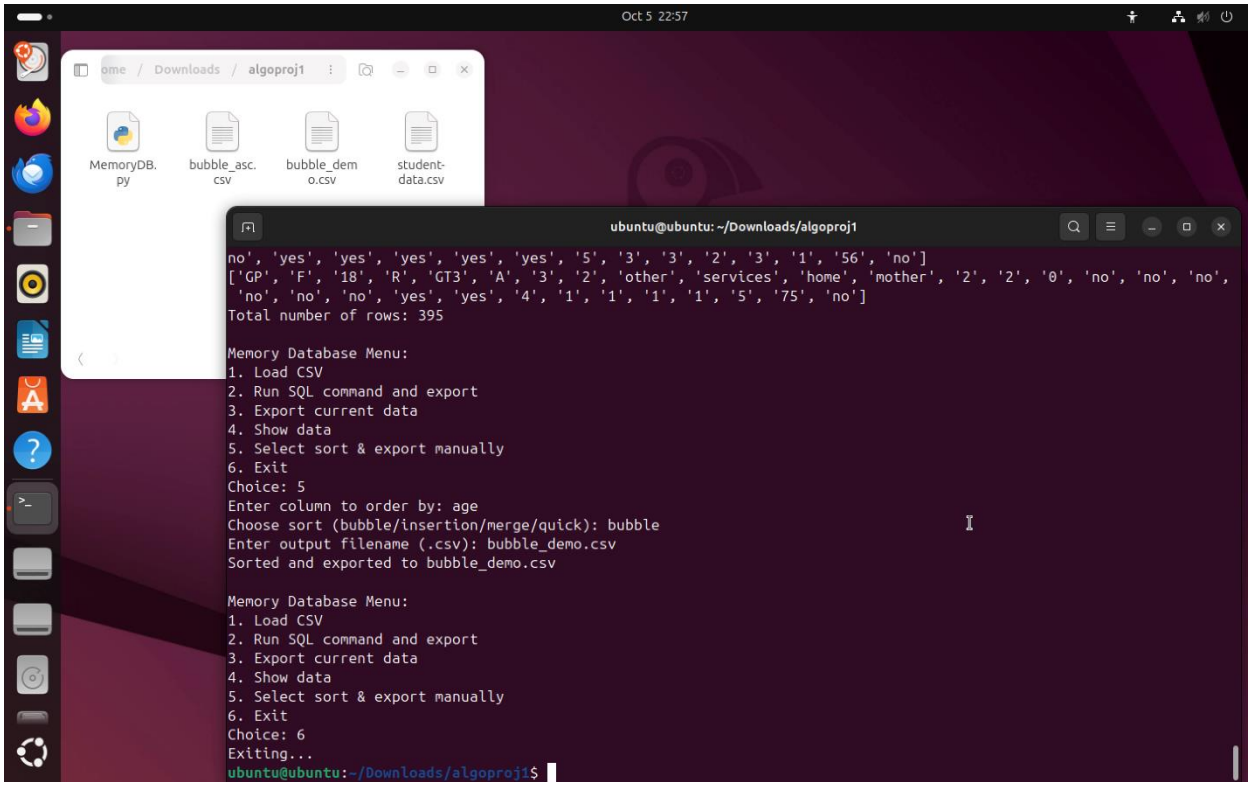

Figure 30 — Bubble sort Python pt. 1

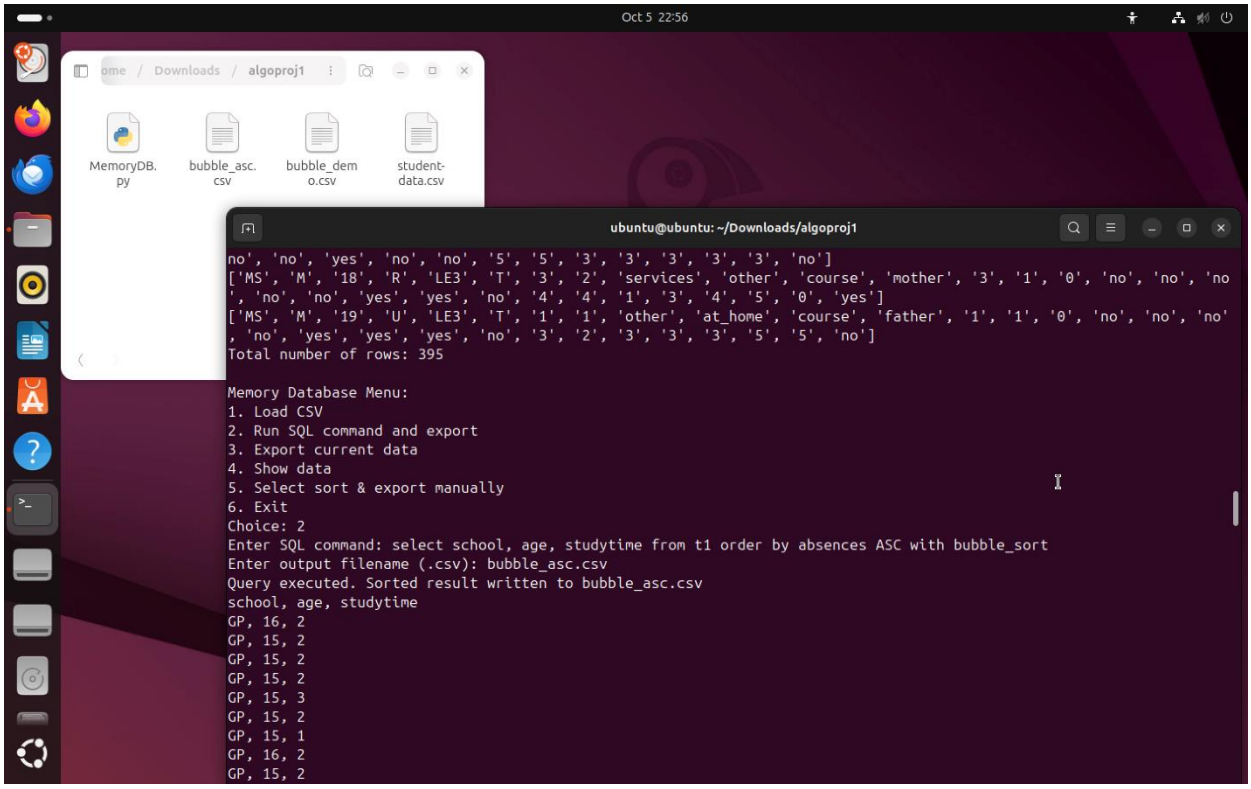

Figure 31 — Bubble sort Python pt. 2

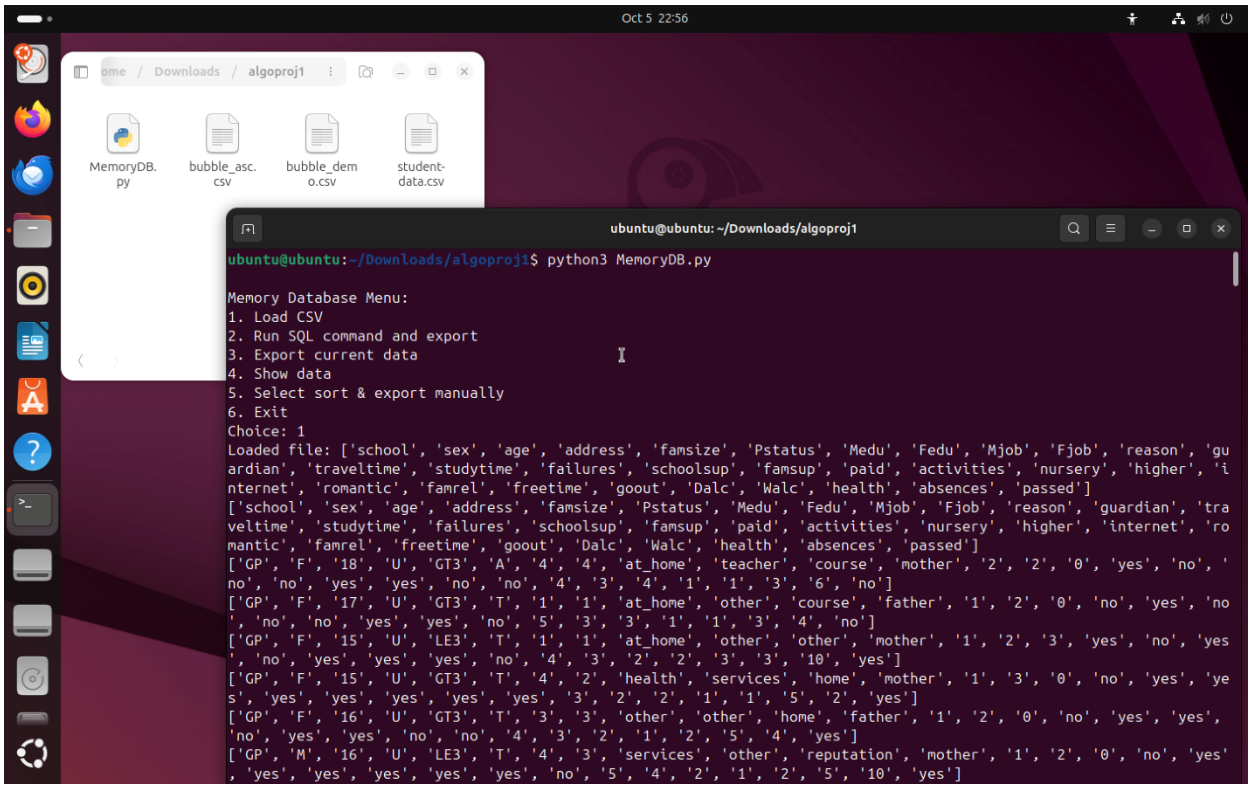

Figure 32 — Bubble sort Python pt. 3

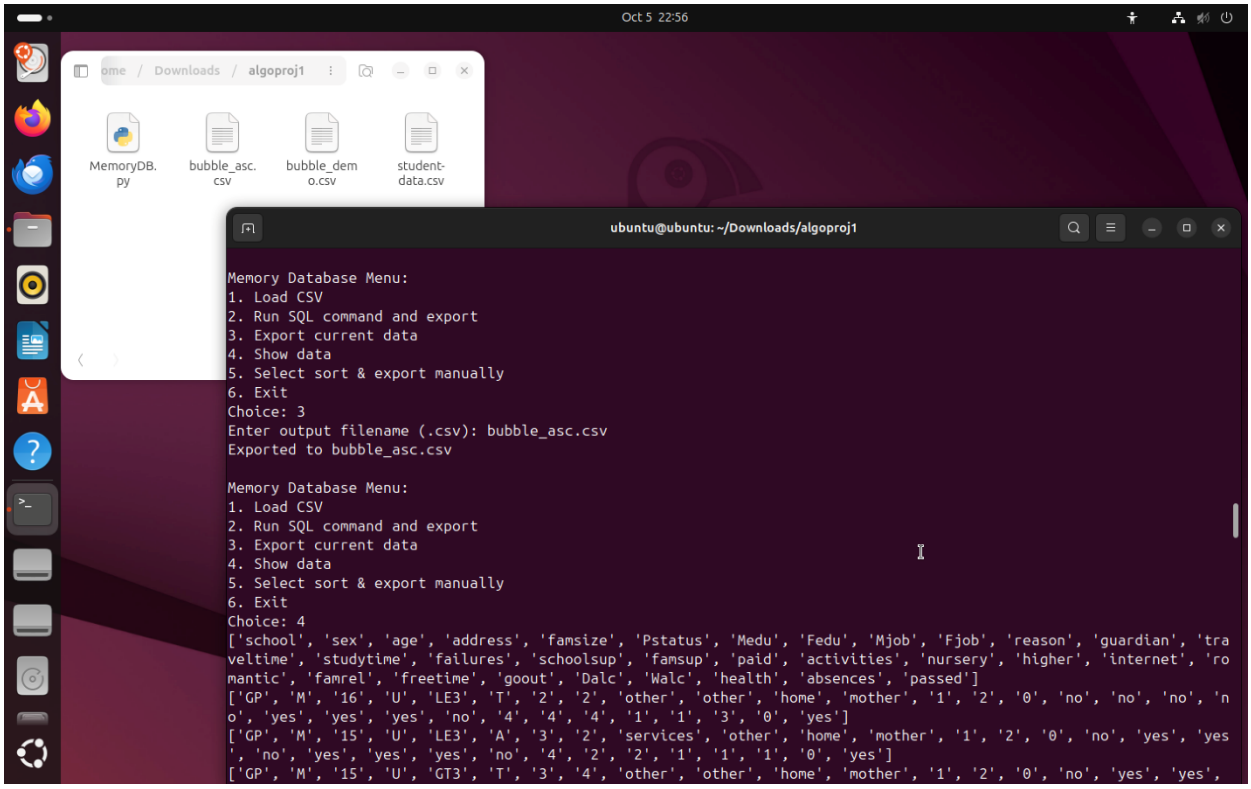

Figure 32 — Bubble sort Python pt. 4

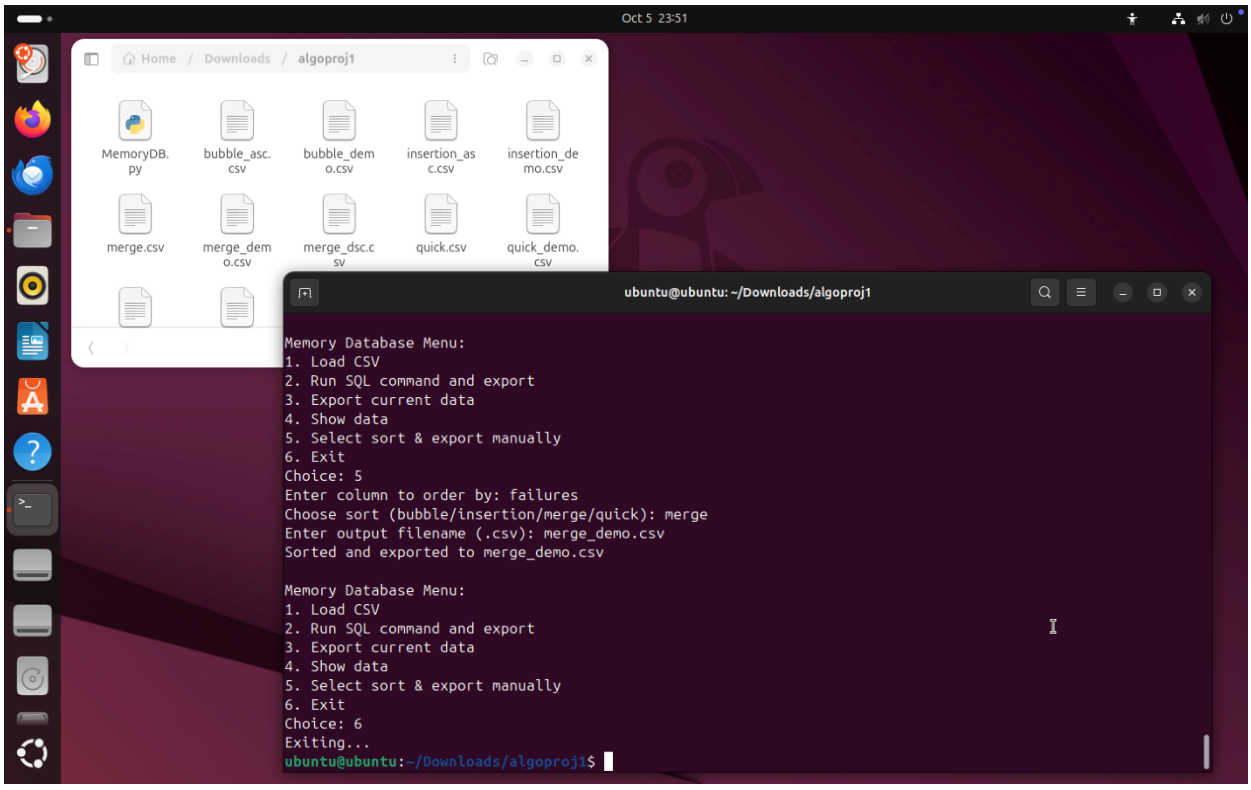

Figure 33 —Merge sort Python pt. 1

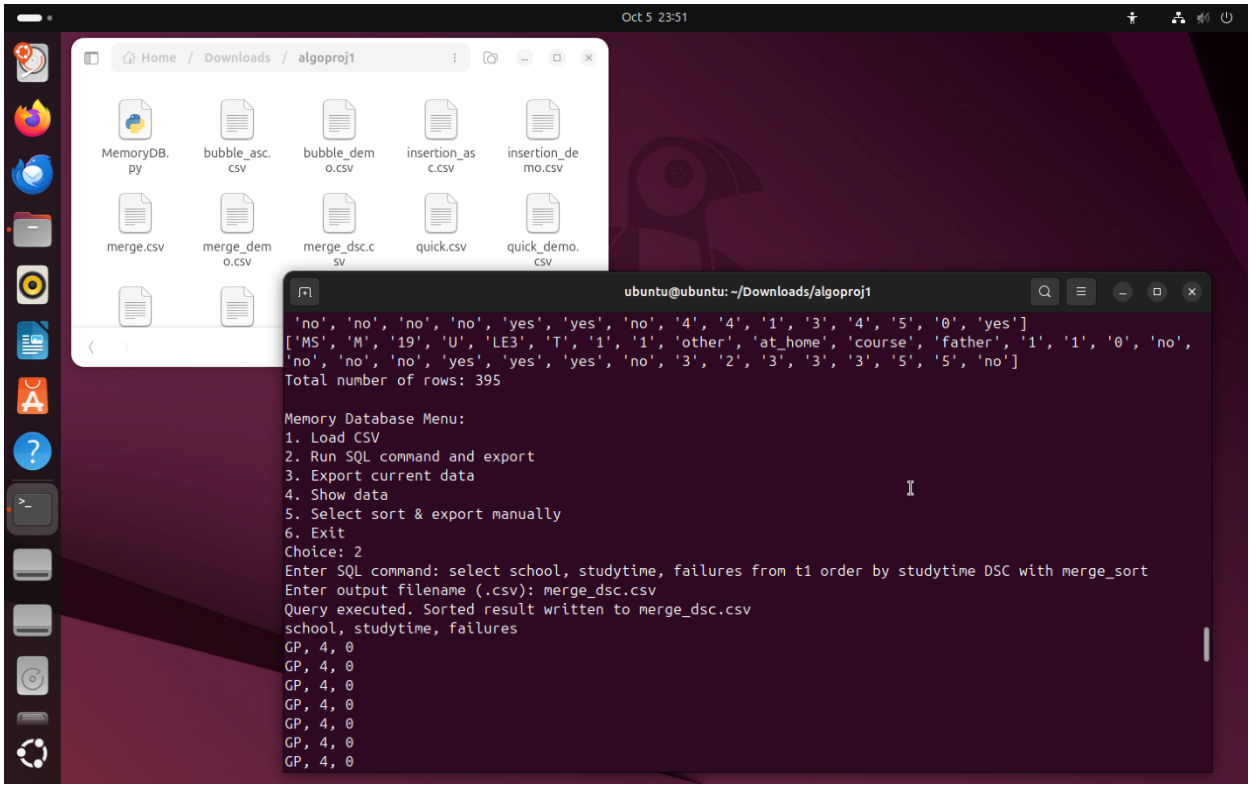

Figure 34 — Merge sort Python pt. 2

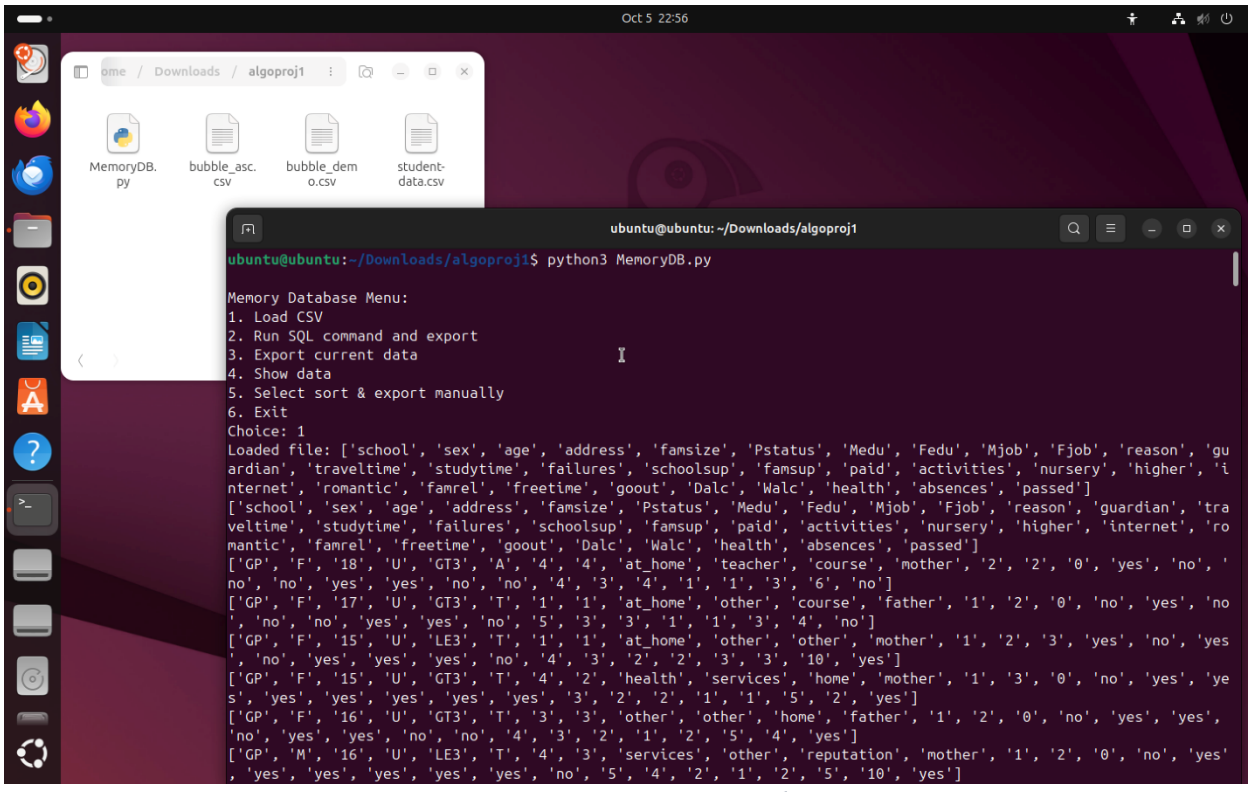

Figure 35 — Merge sort Python pt. 3

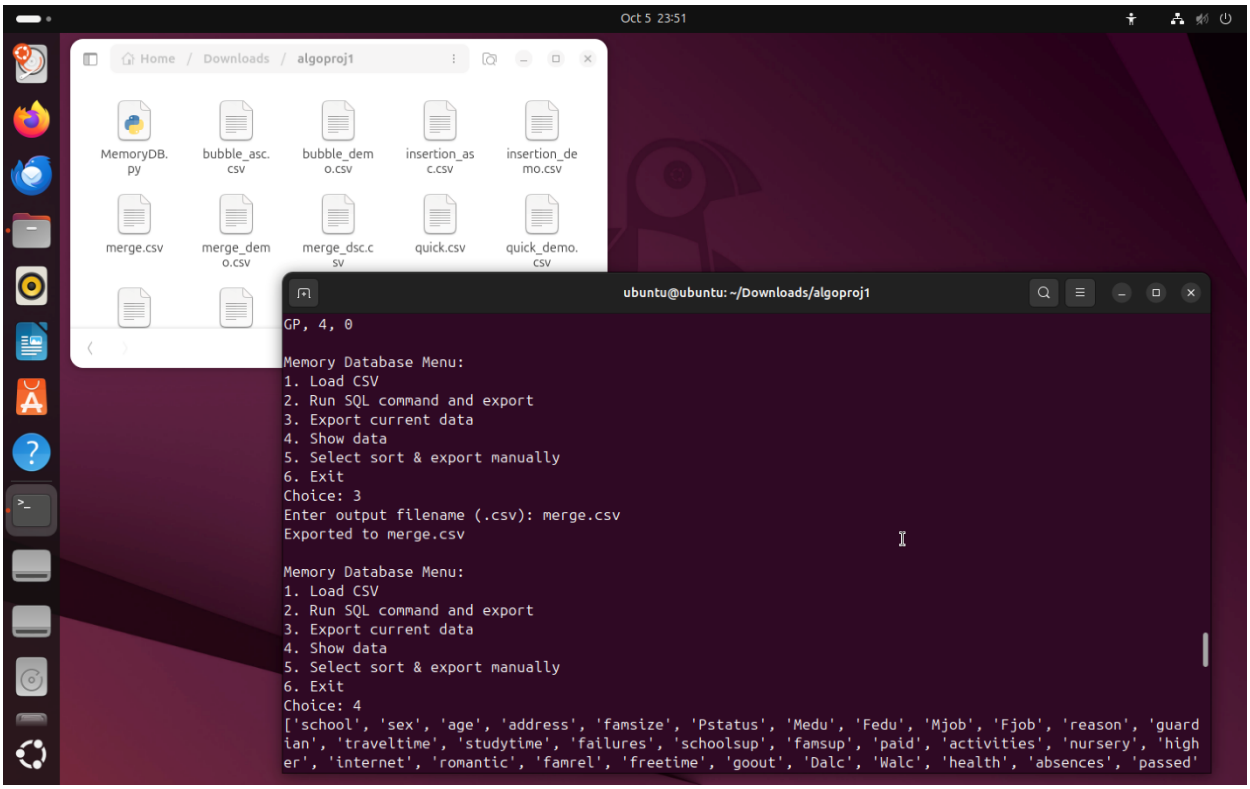

Figure 36 — Merge sort Python pt. 4

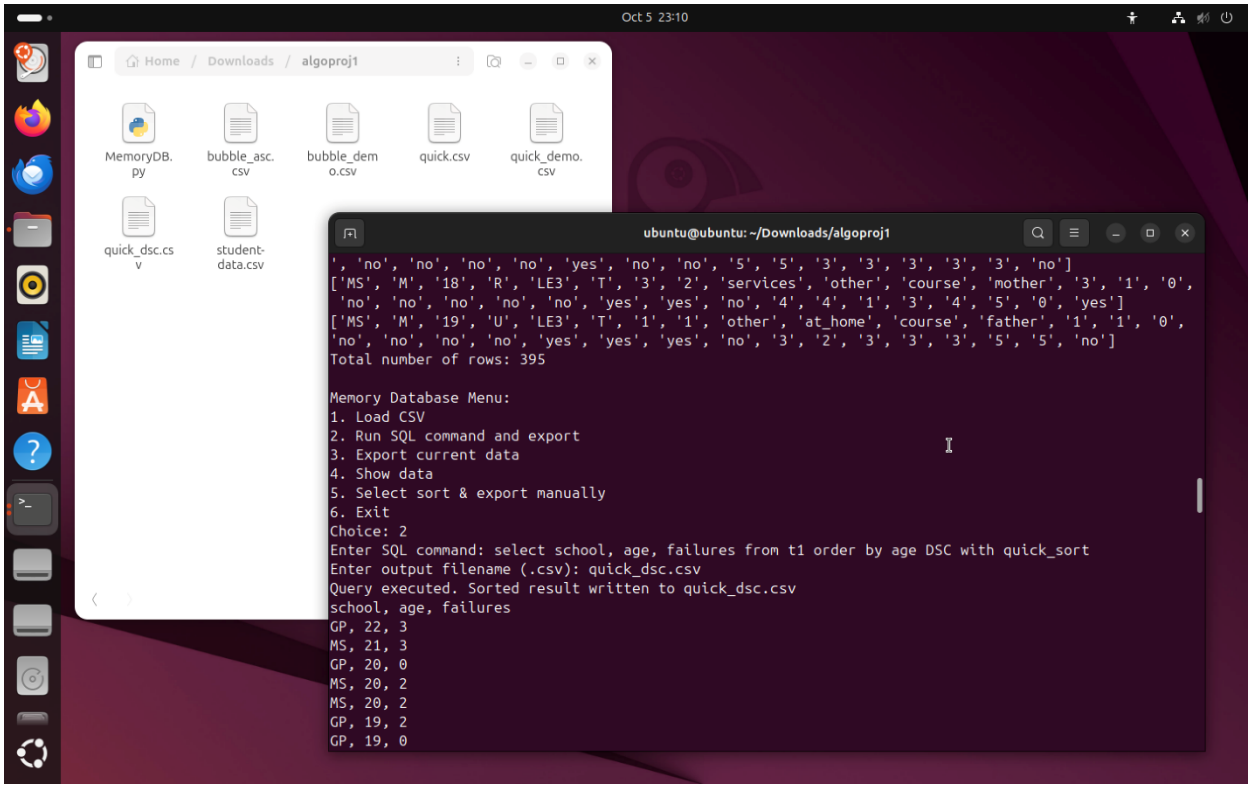

Figure 37 —Quick sort Python pt. 1

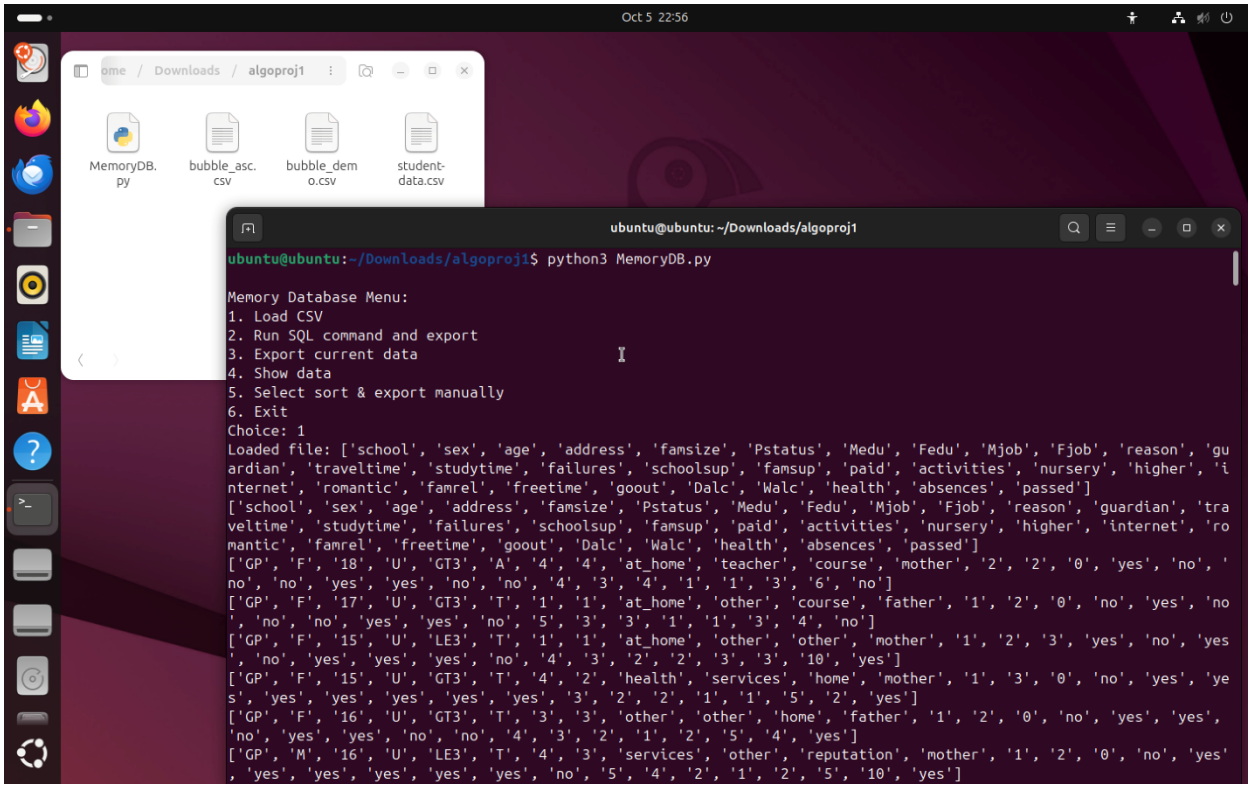

Figure 38 —Quick sort Python pt.2

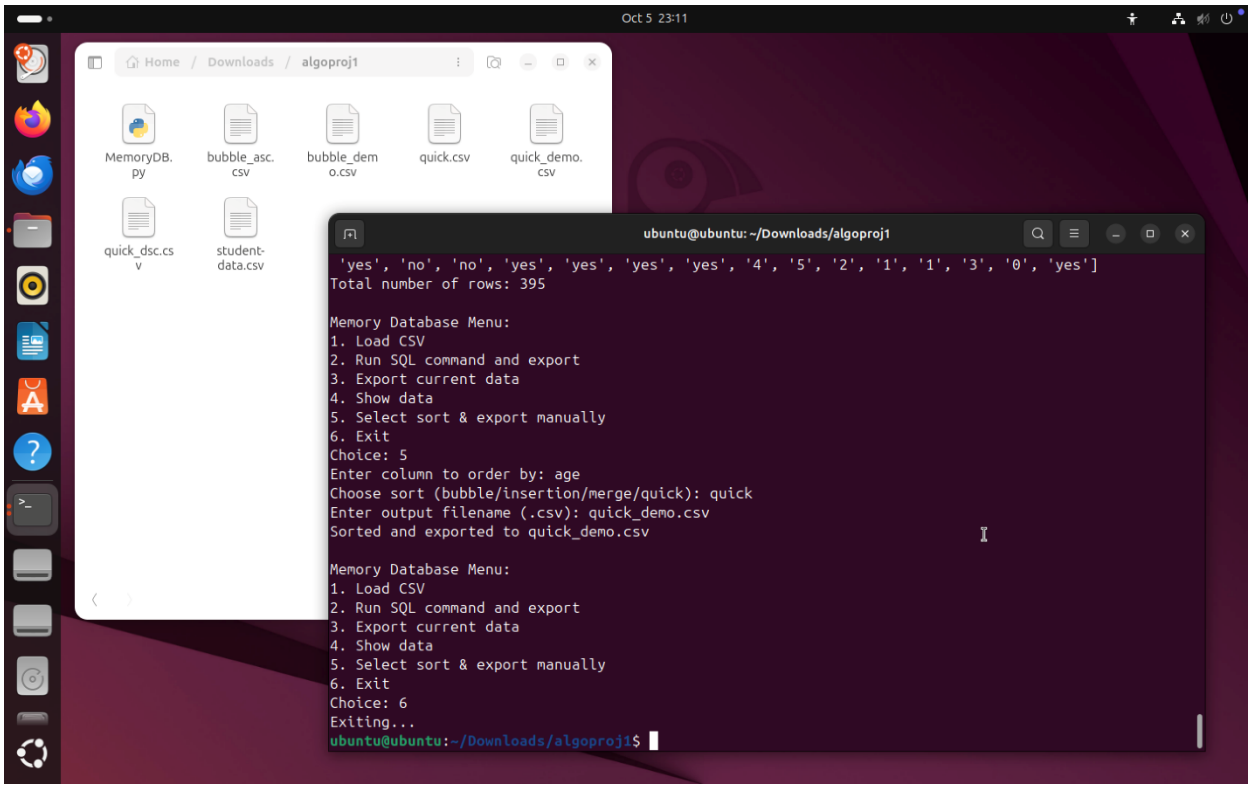

Figure 39 —Quick sort Python pt. 3

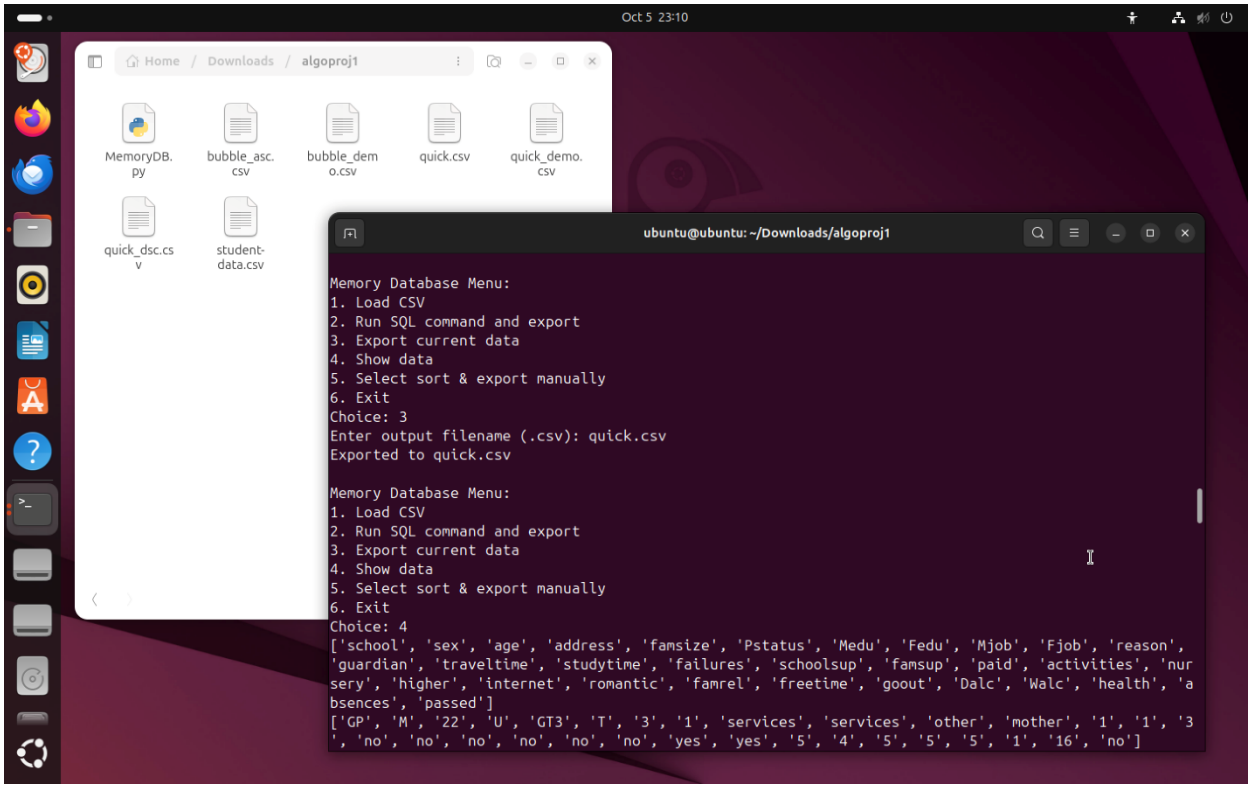

Figure 40 —Quick sort Python pt. 4

V. REPRODUCIBILITY AND CODE ACCESSIBILITY

A foundational principle of computational research is the ability to reproduce and validate results. To ensure the integrity and accessibility of the project's source code, a standardized protocol was established to provide a fully reproducible execution environment for the graders.

This process involved creating public hyperlinks for both the Python and Go implementations. These links are direct to onlinegdb.com, a web-based IDE, which serves as a controlled environment where the code can be executed, and its functionality and correctness can be verified without requiring any installation of any local software. The functionality and correctness can be verified without requiring any local software installation. The source code for both programs is provided below, with a direct link to a runnable version for each:

1. **Python:** https://onlinegdb.com/bMV17hnXx

2. **Java:** https://onlinegdb.com/WA_w1BH9ZU

VI. CONCLUSION

This project successfully demonstrated the design and implementation of a toy in-memory database, built upon linked list structures and extended with multiple sorting algorithms and SQL-like query functionality. Through collaborative effort, we implemented core database features, recursive export functions, and performance benchmarking within controlled environments. The results reinforced fundamental lessons about algorithm efficiency, trade-offs between design simplicity and computational cost, and the practical integration of data structures with system-level profiling. Beyond the technical insights, this project highlighted the value of teamwork, prior knowledge, and structured problem-solving. Collectively, these outcomes provide a strong foundation for approaching more complex data systems and algorithmic challenges in future coursework and professional practice.

ACKNOWLEDGE

The success of this project was a result of the combined effort, knowledge and collaboration of the entire team. Each member played a valuable role in the development of our plan, distribution of the work, and ensure that tasks were completed effectively and in a timely manner. The project benefited greatly from the prior knowledge and technical strengths that everyone brought to the group. Everyone's willingness to share ideas and support one another was essential to our progress.

Through open communication and cooperation, we were able to approach challenges with confidence and develop solutions effectively. The dedication, effort, and professionalism shown by every team member contributed directly to the quality and success of this work.

REFERENCES

1. GDB Online. (n.d.). OnlineGDB: online compiler and debugger. Retrieved November 04, 2025, from https://www.onlinegdb.com/
2. Kelligaki, K. (n.d.). *student-data.csv* Kaggle. Retrieved from https://www.kaggle.com/datasets/kellygakii/student-data-csv

Appendix

1. Java Program

```
import java.io.*;
import java.util.*;

public class MemoryDBHuffman{

   // Memory DB
   static LinkedList<String[]> data = new LinkedList<>();

   // Column indices
   static int ageCol = -1;
```

```
    static int famsizeCol = -1;
    static int absencesCol = -1;

    // Huffman structure
    static class HNode { String token; int freq; HNode l, r; HNode(String t, int f){token = t;
freq=f;} }
    static Map<String,String> huffCode = new LinkedHashMap<>();
    static Map<String,String> rCode = new HashMap<>();
    static Map<String,Integer> freqMap = new LinkedHashMap<>();
    static HNode hRoot = null;
    static String encodedText = "";
    static List<String> decodedTokens = new ArrayList<>();

    // Red-Black Tree Implementation
    enum Color { RED, BLACK }

    static class RBNode {
        int key;
        Color color;
        RBNode left, right, parent;
        RBNode(int k) { key = k; color = Color.RED; left = right = parent = null; }
    }
```

```
// A simple Red-Black Tree supporting insert with step logging
static class RedBlackTree {
    RBNode root = null;
    final List<String> steps = new ArrayList<>();

    void log(String s) {steps.add(s + "\n" + getTreeAsString());}
    void clearLog() { steps.clear(); }
    List<String> getLog() { return new ArrayList<>(steps); }

    void insert(int key) {
        RBNode z = new RBNode(key);
        RBNode y = null;
        RBNode x = root;
        while (x != null) {
            if (key == x.key) {
                return;
            }
            y = x;
            if (z.key < x.key) x = x.left;
            else x = x.right;
        }
        z.parent = y;
        if (y == null) {
```

```
            root = z;
            log("Insert " + key + " as root");
        } else if (z.key < y.key) {
            y.left = z;
            log("Insert " + key + " as left child of " + y.key);
        } else {
            y.right = z;
            log("Insert " + key + " as right child of " + y.key);
        }
        z.left = null;
        z.right = null;
        z.color = Color.RED;
        insertFixup(z);
    }

    private void insertFixup(RBNode z) {
        while (z.parent != null && z.parent.color == Color.RED) {
            RBNode gp = z.parent.parent;
            if (gp == null) break;
            if (z.parent == gp.left) {
                RBNode y = gp.right;
                if (y != null && y.color == Color.RED) {
                    // Case 1: uncle red
```

```
                    z.parent.color = Color.BLACK;
                    y.color = Color.BLACK;
                    gp.color = Color.RED;
                    log("Recoloring: parent " + z.parent.key + " and uncle " + y.key + " to BLACK,
grandparent " + gp.key + " to RED");
                    z = gp;
                } else {
                    if (z == z.parent.right) {
                        // Case 2: left-right
                        z = z.parent;
                        log("Left rotate at " + z.key + " (to handle LR case)");
                        leftRotate(z);
                    }
                    // Case 3: left-left
                    log("Right rotate at " + gp.key + " and recolor parent and grandparent");
                    z.parent.color = Color.BLACK;
                    gp.color = Color.RED;
                    rightRotate(gp);
                }
            } else {
                RBNode y = gp.left;
                if (y != null && y.color == Color.RED) {
                    z.parent.color = Color.BLACK;
```

```
                y.color = Color.BLACK;
                gp.color = Color.RED;
                log("Recoloring: parent " + z.parent.key + " and uncle " + y.key + " to BLACK,
grandparent " + gp.key + " to RED");
                z = gp;
            } else {
                if (z == z.parent.left) {
                    z = z.parent;
                    log("Right rotate at " + z.key + " (to handle RL case)");
                    rightRotate(z);
                }
                log("Left rotate at " + gp.key + " and recolor parent and grandparent");
                z.parent.color = Color.BLACK;
                gp.color = Color.RED;
                leftRotate(gp);
            }
        }
        if (root != null && root.color != Color.BLACK) {
            root.color = Color.BLACK;
            log("Set root color to BLACK");
        }
    }
```

```
private void leftRotate(RBNode x) {
    RBNode y = x.right;
    if (y == null) return;
    x.right = y.left;
    if (y.left != null) y.left.parent = x;
    y.parent = x.parent;
    if (x.parent == null) root = y;
    else if (x == x.parent.left) x.parent.left = y;
    else x.parent.right = y;
    y.left = x;
    x.parent = y;
    log("Performed left-rotate at " + x.key + ", new parent " + y.key);
}

private void rightRotate(RBNode x) {
    RBNode y = x.left;
    if (y == null) return;
    x.left = y.right;
    if (y.right != null) y.right.parent = x;
    y.parent = x.parent;
    if (x.parent == null) root = y;
    else if (x == x.parent.right) x.parent.right = y;
```

```
    else x.parent.left = y;
    y.right = x;
    x.parent = y;
    log("Performed right-rotate at " + x.key + ", new parent " + y.key);
}

// In-order traversal to collect keys
List<Integer> inOrderKeys() {
    List<Integer> keys = new ArrayList<>();
    inOrderRec(root, keys);
    return keys;
}

private void inOrderRec(RBNode n, List<Integer> out) {
    if (n == null) return;
    inOrderRec(n.left, out);
    out.add(n.key);
    inOrderRec(n.right, out);
}

public void printTree(String t) {System.out.println("\n" + t);printHelper(root, "", true); }

private String getTreeAsString() {
```

```
        StringBuilder sb = new StringBuilder();
        buildTreeString(root, "", true, sb);
        return sb.toString();
    }

    private void buildTreeString(RBNode node, String indent, boolean isRight, StringBuilder
sb) {
        if (node == null) return;
        sb.append(indent);
        if (isRight) {
            sb.append("R----");
            indent += "     ";
        } else {
            sb.append("L----");
            indent += "|    ";
        }
        sb.append("(").append(node.key).append(")[").append(node.color == Color.RED ? "Red"
: "Black").append("]\n");
        buildTreeString(node.left, indent, false, sb);
        buildTreeString(node.right, indent, true, sb);
    }
```

```
    private void printHelper(RBNode node, String indent, boolean isRight) {
        if (node == null) return;

        System.out.print(indent);
        if (isRight) {
            System.out.print("R----");
            indent += "     ";
        } else {
            System.out.print("L----");
            indent += "|    ";
        }

        System.out.println("(" + node.key + ")[" + (node.color == Color.RED ? "Red" : "Black")
+ "]");

        printHelper(node.left, indent, false);
        printHelper(node.right, indent, true);
    }
  }

  static RedBlackTree rbAge = new RedBlackTree();
  static RedBlackTree rbAbs = new RedBlackTree();
```

```
// ---------------- CSV Loading ----------------
static void loadCsvRecursive(String filename) {
    data.clear();
    ageCol = famsizeCol = absencesCol = -1;
    File f = new File(filename);
    if (!f.exists()) {
        System.out.println("File not found: " + filename);
        return;
    }

    try (BufferedReader br = new BufferedReader(new FileReader(f))) {
        String header = br.readLine();
        if (header == null) { System.out.println("Empty file"); return; }
        String[] hdr = splitCsvLine(header);

        for (int i = 0; i < hdr.length; i++) {
            String h = hdr[i].trim().toLowerCase();
            if (h.equals("age")) ageCol = i;
            if (h.equals("famsize") || h.equals("family") || h.equals("fam_size")) famsizeCol = i;
            if (h.equals("absences")) absencesCol = i;
        }

        // recursive loading function
```

```
            loadLinesRecursive(br);

            // final fallback
            if (famsizeCol == -1 || ageCol == -1 || absencesCol == -1) {
                if (!data.isEmpty()) {
                    int n = data.getFirst().length;
                    if (famsizeCol == -1) famsizeCol = Math.max(0, n-3);
                    if (ageCol == -1) ageCol = Math.max(0, n-2);
                    if (absencesCol == -1) absencesCol = Math.max(0, n-1);
                } else { famsizeCol = ageCol = absencesCol = 0; }
            }

            System.out.println("Loaded " + data.size() + " rows. Using indices -> age:" + ageCol + ",
famsize:" + famsizeCol + ", absences:" + absencesCol);

        } catch (IOException e) {
            System.err.println("Error reading: " + e.getMessage());
        }

        // reset Huffman and trees
        freqMap.clear(); huffCode.clear(); rCode.clear(); hRoot=null; encodedText="";
decodedTokens.clear();
        rbAge = new RedBlackTree(); rbAbs = new RedBlackTree();
```

```
    }

    private static void loadLinesRecursive(BufferedReader br) throws IOException {
        String line = br.readLine();
        if (line == null) return; // base case
        if (!line.trim().isEmpty()) {
            String[] parts = splitCsvLine(line);
            data.add(parts);

            // fallback detection for age/absences/famsize
            if (ageCol == -1 || absencesCol == -1 || famsizeCol == -1) {
                for (int i = 0; i < parts.length; i++) {
                    String p = parts[i].trim();
                    if (ageCol == -1) {
                        try { int v = Integer.parseInt(p); if (v>=5 && v<=120) ageCol = i; }
catch(Exception ex){}
                    }
                    if (absencesCol == -1) {
                        try { int v = Integer.parseInt(p); if (v>=0 && v<=500) absencesCol = i; }
catch(Exception ex){}
                    }
                }
            }
```

```
    }
    loadLinesRecursive(br);
}



// CSV split helper
static String[] splitCsvLine(String line) {
    List<String> out = new ArrayList<>();
    StringBuilder cur = new StringBuilder();
    boolean inQ = false;
    for (int i=0;i<line.length();i++) {
        char c = line.charAt(i);
        if (c == '"') { inQ = !inQ; continue; }
        if (c==',' && !inQ) { out.add(cur.toString()); cur.setLength(0); }
        else cur.append(c);
    }
    out.add(cur.toString());
    return out.toArray(new String[0]);
}


// ---------- Huffman ----------
static String makeToken(String[] row) {
    String fam = (famsizeCol>=0 && famsizeCol<row.length) ? row[famsizeCol].trim() : "";
```

```
        String age = (ageCol>=0 && ageCol<row.length) ? row[ageCol].trim() : "";
        return fam + age;
    }

    static void buildFrequency() {
        freqMap.clear();
        for (String[] row : data) {
            String t = makeToken(row);
            freqMap.put(t, freqMap.getOrDefault(t,0)+1);
        }
    }

    static void buildHuffmanTree() {
        hRoot = null;
        if (freqMap.isEmpty()) return;
        PriorityQueue<HNode> pq = new PriorityQueue<>(Comparator.comparingInt(a->a.freq));
        for (Map.Entry<String,Integer> e: freqMap.entrySet()) pq.add(new HNode(e.getKey(),
e.getValue()));
        if (pq.size()==1) {
            HNode only = pq.poll();
            hRoot = new HNode(null, only.freq);
            hRoot.l = only;
        } else {
```

```
        while (pq.size()>1) {
            HNode a = pq.poll(); HNode b = pq.poll();
            HNode p = new HNode(null, a.freq + b.freq);
            p.l = a; p.r = b;
            pq.add(p);
        }
        hRoot = pq.poll();
    }
}

static void generateCodes() {
    huffCode.clear(); rCode.clear();
    genRec(hRoot, "");
}
static void genRec(HNode n, String code) {
    if (n==null) return;
    if (n.l==null && n.r==null) {
        String tk = n.token;
        if (code.length()==0) code="0";
        huffCode.put(tk, code);
        rCode.put(code, tk);
        return;
    }
```

```
        genRec(n.l, code + "0");
        genRec(n.r, code + "1");
    }

    static void writeDictionaryCsv(String fname) {
        try (BufferedWriter bw = new BufferedWriter(new FileWriter(fname))) {
            bw.write("Token,Frequency,Code,Bits"); bw.newLine();
            for (Map.Entry<String,String> e : huffCode.entrySet()) {
                String tok = e.getKey(); String code = e.getValue();
                int f = freqMap.getOrDefault(tok,0);
                bw.write("\"" + tok.replace("\"","\"\"") + "\"," + f + "," + code + "," + code.length());
bw.newLine();
            }
            System.out.println("Wrote dictionary.csv");
        } catch (IOException ex) { System.err.println("Write error: " + ex.getMessage()); }
    }

    static void encodeData() {
        if (huffCode.isEmpty()) { System.out.println("No codes generated"); return; }
        StringBuilder sb = new StringBuilder();
        for (String[] row : data) {
            String t = makeToken(row);
            String code = huffCode.get(t);
```

```
        if (code==null) { System.err.println("Missing code for " + t); } else sb.append(code);
    }
    encodedText = sb.toString();
    try (BufferedWriter bw = new BufferedWriter(new FileWriter("encoded_text.txt"))) {
        bw.write(encodedText);
        System.out.println("Wrote encoded_text.txt (len=" + encodedText.length() + " bits)");
    } catch (IOException e) { System.err.println("Encode write error: " + e.getMessage()); }
}

static void decodeData() {
    decodedTokens.clear();
    if (encodedText==null || encodedText.isEmpty()) {
        File f = new File("encoded_text.txt");
        if (f.exists()) encodedText = readFile(f.getPath());
        else { System.out.println("No encoded text present"); return; }
    }
    StringBuilder tmp = new StringBuilder();
    for (char c : encodedText.toCharArray()) {
        tmp.append(c);
        String s = tmp.toString();
        if (rCode.containsKey(s)) {
            decodedTokens.add(rCode.get(s));
            tmp.setLength(0);
```

```
        }
    }
    try (BufferedWriter bw = new BufferedWriter(new FileWriter("decoded_text.txt"))) {
        for (String tok : decodedTokens) { bw.write(tok); bw.newLine(); }
        System.out.println("Wrote decoded_text.txt (count=" + decodedTokens.size() + ")");
    } catch (IOException e) { System.err.println("Decoded write error: " + e.getMessage()); }
}

static int computeOriginalBits() {
    int sum = 0;
    for (String[] row : data) sum += makeToken(row).length() * 8;
    return sum;
}

static int computeCompressedBits() {
    int s = 0;
    for (String[] row : data) {
        String code = huffCode.get(makeToken(row));
        if (code!=null) s += code.length();
    }
    return s;
}
```

```
   static int gcd(int a,int b){ a=Math.abs(a); b=Math.abs(b); if (b==0) return a==0?1:a; return
gcd(b,a%b); }


   // ---------------- Menu ----------------
   static void menuLoop() {
      Scanner sc = new Scanner(System.in);
      while (true) {
         System.out.println("\n--- Menu ---");
         System.out.println("1 Load CSV into LinkedList");
         System.out.println("2 Display Data");
         System.out.println("3 Generate Huffman Codes");
         System.out.println("4 Show Huffman Dictionary & stats");
         System.out.println("5 Encode Data ");
         System.out.println("6 Decode Encoded Text");
         System.out.println("7 Build Red-Black Tree Index (age & absences) with logs");
         System.out.println("8 Display RBT insertion logs and trees");
         System.out.println("9 Exit");
         System.out.print("Choice: ");
         String choice = sc.nextLine().trim();
         if (choice.isEmpty()) continue;
         try {
            int c = Integer.parseInt(choice);
            switch (c) {
```

```
                case 1:
                    System.out.print("CSV filename (default student-data.csv): ");
                    String fn = sc.nextLine().trim(); if (fn.isEmpty()) fn="student-data.csv";
                    loadCsvRecursive(fn); break;
                case 2:
                    showData(50); break;
                case 3:
                    if (data.isEmpty()) { System.out.println("Load CSV first."); break; }
                    buildFrequency(); buildHuffmanTree(); generateCodes();
                    System.out.println("Huffman codes generated. Tokens: " + huffCode.size());
break;
                case 4:
                    if (huffCode.isEmpty()) { System.out.println("Generate codes first."); break; }
                    printDictionary(); writeDictionaryCsv("dictionary.csv");
                    int orig = computeOriginalBits(), comp = computeCompressedBits();
                    System.out.println("Original Size (bits): " + orig);
                    System.out.println("Compressed Size (bits): " + comp);
                    int g = gcd(orig, comp);
                    System.out.println("Compression Ratio: " + (orig/g) + ":" + (comp/g) + "
(decimal: " + String.format("%.2f", (double)orig/comp) + ")");
                    break;
                case 5:
                    if (huffCode.isEmpty()) { System.out.println("Generate codes first."); break; }
```

```
            encodeData(); break;
        case 6:
            if (huffCode.isEmpty()) { System.out.println("Generate codes first."); break; }
            decodeData(); break;
        case 7:
            if (data.isEmpty()) { System.out.println("Load CSV first."); break; }
            // build RBT by inserting keys in order and logging steps
            rbAge = new RedBlackTree(); rbAbs = new RedBlackTree();
            rbAge.clearLog(); rbAbs.clearLog();
            for (String[] row : data) {
                int a = safeInt(row, ageCol);
                int abs = safeInt(row, absencesCol);
                rbAge.insert(a);
                rbAbs.insert(abs);
            }
            System.out.println("Built RBTs for age and absences. Logs recorded.");
            break;
        case 8:
            System.out.println("\n--- Age RBT Logs ---");
            List<String> logsA = rbAge.getLog();
            if (logsA.isEmpty()) System.out.println("[No logs - build RBT first]");
            else {
                for (String s : logsA) {
```

```
                System.out.println(s);
                System.out.println();
            }
        }
        System.out.println("\nAge tree keys (in-order): " + rbAge.inOrderKeys());
        rbAge.printTree("Age Final Red Black Tree Structure:");
        System.out.println("\n--- Absences RBT Logs ---");
        List<String> logsB = rbAbs.getLog();
        if (logsB.isEmpty()) System.out.println("[No logs - build RBT first]");
        else {
            for (String s : logsB){
                System.out.println(s);
                System.out.println();
            }
        }
        System.out.println("\nAbsences tree keys (in-order): " + rbAbs.inOrderKeys());
        rbAbs.printTree("Absences Final Red Black Tree Structure:");
        break;
    case 9:
        System.out.println("Exit."); sc.close(); return;
    default:
        System.out.println("Invalid choice.");
}
```

```
        } catch (NumberFormatException nfe) {
            System.out.println("Enter a number.");
        } catch (Exception ex) {
            System.err.println("Error: " + ex.getMessage()); ex.printStackTrace(System.err);
        }
    }
}

static int safeInt(String[] row, int idx) {
    if (idx < 0 || idx >= row.length) return 0;
    try { return Integer.parseInt(row[idx].trim()); } catch(Exception ex) { return 0; }
}

static void showData(int limit) {
    if (data.isEmpty()) { System.out.println("[No data]"); return; }
    int c = 0;
    for (String[] r : data) {
        System.out.println(Arrays.toString(r));
        if (++c >= limit) break;
    }
    System.out.println("Displayed " + Math.min(limit, data.size()) + " of " + data.size());
}
```

```
    static void printDictionary() {
        System.out.printf("%-20s %-8s %-15s %-5s%n", "Token", "Freq", "Code", "Bits");
        System.out.println("----------------------------------------------------------");
        for (Map.Entry<String,String> e : huffCode.entrySet()) {
            String t = e.getKey(); String code = e.getValue();
            System.out.printf("%-20s %-8d %-15s %-5d%n", t, freqMap.getOrDefault(t,0), code,
code.length());
        }
    }



    static String readFile(String fname) {
        StringBuilder sb = new StringBuilder();
        try (BufferedReader br = new BufferedReader(new FileReader(fname))) {
            String l; while ((l=br.readLine())!=null) sb.append(l.trim());
        } catch (IOException e) { System.err.println("Read error: " + e.getMessage()); }
        return sb.toString();
    }
    // ---------------- main ----------------
    public static void main(String[] args) {
        menuLoop();
    }
}
```

2. Python Program – main.py

```
from __future__ import annotations

import sys

import os

from typing import List, Dict, Optional, Tuple

from dataclasses import dataclass

import heapq


# ============================

# In-memory "DB"

# ============================

data: List[List[str]] = []


# Column indices

ageCol: int = -1

famsizeCol: int = -1

absencesCol: int = -1


# ============================

# Huffman structures

# ============================


@dataclass
```

```
class HNode:
    token: Optional[str]
    freq: int
    l: Optional["HNode"] = None
    r: Optional["HNode"] = None

huffCode: Dict[str, str] = {}
rCode: Dict[str, str] = {}
freqMap: Dict[str, int] = {}
hRoot: Optional[HNode] = None
encodedText: str = ""
decodedTokens: List[str] = []

# ========================
# Red-Black Tree structures
# ========================

class Color:
    RED = "RED"
    BLACK = "BLACK"

class RBNode:
    __slots__ = ("key", "color", "left", "right", "parent")
```

```
    def __init__(self, k: int):
        self.key: int = k
        self.color: str = Color.RED
        self.left: Optional["RBNode"] = None
        self.right: Optional["RBNode"] = None
        self.parent: Optional["RBNode"] = None


class RedBlackTree:
    def __init__(self):
        self.root: Optional[RBNode] = None
        self.steps: List[str] = []

    def log(self, s: str):
        self.steps.append(s + "\n" + self.getTreeAsString())

    def clearLog(self):
        self.steps.clear()

    def getLog(self) -> List[str]:
        return list(self.steps)

    def insert(self, key: int):
        z = RBNode(key)
```

```
    y = None
    x = self.root
    while x is not None:
        if key == x.key:
            return  # duplicate: ignore (matches Java)
        y = x
        if z.key < x.key:
            x = x.left
        else:
            x = x.right
    z.parent = y
    if y is None:
        self.root = z
        self.log(f"Insert {key} as root")
    elif z.key < y.key:
        y.left = z
        self.log(f"Insert {key} as left child of {y.key}")
    else:
        y.right = z
        self.log(f"Insert {key} as right child of {y.key}")
    z.left = None
    z.right = None
    z.color = Color.RED
```

```
        self.insertFixup(z)


    def insertFixup(self, z: RBNode):
        while z.parent is not None and z.parent.color == Color.RED:
            gp = z.parent.parent
            if gp is None:
                break
            if z.parent == gp.left:
                y = gp.right
                if y is not None and y.color == Color.RED:
                    z.parent.color = Color.BLACK
                    y.color = Color.BLACK
                    gp.color = Color.RED
                    self.log(f"Recoloring: parent {z.parent.key} and uncle {y.key} to BLACK,
grandparent {gp.key} to RED")
                    z = gp
                else:
                    if z == z.parent.right:
                        z = z.parent
                        self.log(f"Left rotate at {z.key} (to handle LR case)")
                        self.leftRotate(z)
                    self.log(f"Right rotate at {gp.key} and recolor parent and grandparent")
                    z.parent.color = Color.BLACK
```

```
                gp.color = Color.RED
                self.rightRotate(gp)
            else:
                y = gp.left
                if y is not None and y.color == Color.RED:
                    z.parent.color = Color.BLACK
                    y.color = Color.BLACK
                    gp.color = Color.RED
                    self.log(f"Recoloring: parent {z.parent.key} and uncle {y.key} to BLACK,
grandparent {gp.key} to RED")
                    z = gp
                else:
                    if z == z.parent.left:
                        z = z.parent
                        self.log(f"Right rotate at {z.key} (to handle RL case)")
                        self.rightRotate(z)
                    self.log(f"Left rotate at {gp.key} and recolor parent and grandparent")
                    z.parent.color = Color.BLACK
                    gp.color = Color.RED
                    self.leftRotate(gp)
        if self.root is not None and self.root.color != Color.BLACK:
            self.root.color = Color.BLACK
            self.log("Set root color to BLACK")
```

```
def leftRotate(self, x: RBNode):
    y = x.right
    if y is None:
        return
    x.right = y.left
    if y.left is not None:
        y.left.parent = x
    y.parent = x.parent
    if x.parent is None:
        self.root = y
    elif x == x.parent.left:
        x.parent.left = y
    else:
        x.parent.right = y
    y.left = x
    x.parent = y
    self.log(f"Performed left-rotate at {x.key}, new parent {y.key}")

def rightRotate(self, x: RBNode):
    y = x.left
    if y is None:
        return
```

```
    x.left = y.right
    if y.right is not None:
        y.right.parent = x
    y.parent = x.parent
    if x.parent is None:
        self.root = y
    elif x == x.parent.right:
        x.parent.right = y
    else:
        x.parent.left = y
    y.right = x
    x.parent = y
    self.log(f"Performed right-rotate at {x.key}, new parent {y.key}")


def inOrderKeys(self) -> List[int]:
    out: List[int] = []
    def rec(n: Optional[RBNode]):
        if n is None: return
        rec(n.left)
        out.append(n.key)
        rec(n.right)
    rec(self.root)
    return out
```

```
def printTree(self, title: str):
    print("\n" + title)
    self._printHelper(self.root, "", True)

def _printHelper(self, node: Optional[RBNode], indent: str, isRight: bool):
    if node is None:
        return
    print(indent, end="")
    if isRight:
        print("R----", end="")
        indent += "     "
    else:
        print("L----", end="")
        indent += "|    "
    print(f"({node.key})[{'Red' if node.color == Color.RED else 'Black'}]")
    self._printHelper(node.left, indent, False)
    self._printHelper(node.right, indent, True)

def getTreeAsString(self) -> str:
    sb: List[str] = []
    def build(node: Optional[RBNode], indent: str, isRight: bool):
        if node is None:
```

```
            return
        sb.append(indent)
        if isRight:
            sb.append("R----")
            indent2 = indent + "     "
        else:
            sb.append("L----")
            indent2 = indent + "|    "
        sb.append(f"({node.key})[{'Red' if node.color == Color.RED else 'Black'}]\n")
        build(node.left, indent2, False)
        build(node.right, indent2, True)
    build(self.root, "", True)
    return "".join(sb)

rbAge = RedBlackTree()
rbAbs = RedBlackTree()

# ============================
# CSV loading (recursive)
# ============================

def splitCsvLine(line: str) -> List[str]:
    out: List[str] = []
```

```
    cur: List[str] = []
    inQ = False
    i = 0
    while i < len(line):
        c = line[i]
        if c == '"':
            inQ = not inQ
            i += 1
            continue
        if c == ',' and not inQ:
            out.append("".join(cur))
            cur = []
        else:
            cur.append(c)
        i += 1
    out.append("".join(cur))
    return out


def loadCsvRecursive(filename: str):
    global data, ageCol, famsizeCol, absencesCol
    data.clear()
    ageCol = -1
    famsizeCol = -1
```

```
absencesCol = -1
if not os.path.exists(filename):
    print(f"File not found: {filename}")
    _reset_huffman_and_trees()
    return
try:
    with open(filename, "r", encoding="utf-8") as f:
        # header
        header = f.readline()
        if header is None or header == "":
            print("Empty file")
            _reset_huffman_and_trees()
            return
        hdr = splitCsvLine(header.rstrip("\n"))
        # detect indices by header
        for i, h in enumerate(hdr):
            hh = h.strip().lower()
            if hh == "age":
                ageCol = i
            if hh in ("famsize", "family", "fam_size"):
                famsizeCol = i
            if hh == "absences":
                absencesCol = i
```

```
# recursive load of lines
def _load_lines_recursive(fileobj):
    line = fileobj.readline()
    if not line:
        return
    if line.strip():
        parts = splitCsvLine(line.rstrip("\n"))
        data.append(parts)
        # fallback detect for age/absences while reading first rows
        global ageCol, absencesCol, famsizeCol
        # (match Java: only infer age/abs if unknown)
        if ageCol == -1 or absencesCol == -1 or famsizeCol == -1:
            for idx, p in enumerate(parts):
                p2 = p.strip()
                if ageCol == -1:
                    try:
                        v = int(p2)
                        if 5 <= v <= 120:
                            ageCol = idx
                    except:
                        pass
                if absencesCol == -1:
```

```
                    try:
                        v = int(p2)
                        if 0 <= v <= 500:
                            absencesCol = idx
                    except:
                        pass
            _load_lines_recursive(fileobj)

        _load_lines_recursive(f)

        # final fallback if any missing
        if famsizeCol == -1 or ageCol == -1 or absencesCol == -1:
            if data:
                n = len(data[0])
                if famsizeCol == -1: famsizeCol = max(0, n - 3)
                if ageCol == -1: ageCol = max(0, n - 2)
                if absencesCol == -1: absencesCol = max(0, n - 1)
            else:
                famsizeCol = ageCol = absencesCol = 0

        print(f'Loaded {len(data)} rows. Using indices -> age:{ageCol}, famsize:{famsizeCol},
absences:{absencesCol}")
    except Exception as e:
```

```
        print(f"Error reading: {e}")


    _reset_huffman_and_trees()


def _reset_huffman_and_trees():
    global freqMap, huffCode, rCode, hRoot, encodedText, decodedTokens, rbAge, rbAbs
    freqMap.clear()
    huffCode.clear()
    rCode.clear()
    hRoot = None
    encodedText = ""
    decodedTokens.clear()
    rbAge = RedBlackTree()
    rbAbs = RedBlackTree()


# ==========================
# Huffman logic
# ==========================


def makeToken(row: List[str]) -> str:
    fam = row[famsizeCol].strip() if 0 <= famsizeCol < len(row) else ""
    age = row[ageCol].strip() if 0 <= ageCol < len(row) else ""
    return f"{fam}{age}"
```

```
def buildFrequency():
    freqMap.clear()
    for row in data:
        t = makeToken(row)
        freqMap[t] = freqMap.get(t, 0) + 1


def buildHuffmanTree():
    global hRoot
    hRoot = None
    if not freqMap:
        return
    # min-heap by freq
    heap: List[Tuple[int, int, HNode]] = []
    uid = 0
    for tok, fr in freqMap.items():
        heapq.heappush(heap, (fr, uid, HNode(tok, fr)))
        uid += 1
    if len(heap) == 1:
        _, _, only = heapq.heappop(heap)
        hRoot = HNode(None, only.freq)
        hRoot.l = only
        return
```

```
    while len(heap) > 1:
        f1, _, a = heapq.heappop(heap)
        f2, _, b = heapq.heappop(heap)
        p = HNode(None, f1 + f2, a, b)
        heapq.heappush(heap, (p.freq, uid, p))
        uid += 1
    hRoot = heapq.heappop(heap)[2]


def generateCodes():
    huffCode.clear()
    rCode.clear()
    def genRec(n: Optional[HNode], code: str):
        if n is None:
            return
        if n.l is None and n.r is None:
            tk = n.token
            c = code if len(code) > 0 else "0"
            huffCode[tk] = c
            rCode[c] = tk
            return
        genRec(n.l, code + "0")
        genRec(n.r, code + "1")
    genRec(hRoot, "")
```

```
def writeDictionaryCsv(fname: str):
    try:
        with open(fname, "w", encoding="utf-8", newline="") as bw:
            bw.write("Token,Frequency,Code,Bits\n")
            for tok, code in huffCode.items():
                f = freqMap.get(tok, 0)
                safe_tok = tok.replace("", """)
                bw.write(f"\"{safe_tok}\",{f},{code},{len(code)}\n")
        print("Wrote dictionary.csv")
    except Exception as ex:
        print(f"Write error: {ex}")


def encodeData():
    global encodedText
    if not huffCode:
        print("No codes generated")
        return
    sb: List[str] = []
    for row in data:
        t = makeToken(row)
        code = huffCode.get(t)
        if code is None:
```

```
            print(f"Missing code for {t}", file=sys.stderr)
        else:
            sb.append(code)
    encodedText = "".join(sb)
    try:
        with open("encoded_text.txt", "w", encoding="utf-8") as bw:
            bw.write(encodedText)
        print(f"Wrote encoded_text.txt (len={len(encodedText)} bits)")
    except Exception as e:
        print(f"Encode write error: {e}", file=sys.stderr)


def decodeData():
    global decodedTokens, encodedText
    decodedTokens.clear()
    if not encodedText:
        if os.path.exists("encoded_text.txt"):
            encodedText = readFile("encoded_text.txt")
        else:
            print("No encoded text present")
            return
    tmp: List[str] = []
    cur = ""
    # Simple prefix-matching using rCode (same as Java loop)
```

```
    for ch in encodedText:
        tmp.append(ch)
        cur = "".join(tmp)
        if cur in rCode:
            decodedTokens.append(rCode[cur])
            tmp.clear()
            cur = ""
    try:
        with open("decoded_text.txt", "w", encoding="utf-8") as bw:
            for tok in decodedTokens:
                bw.write(tok + "\n")
        print(f"Wrote decoded_text.txt (count={len(decodedTokens)})")
    except Exception as e:
        print(f"Decoded write error: {e}", file=sys.stderr)


def computeOriginalBits() -> int:
    s = 0
    for row in data:
        s += len(makeToken(row)) * 8
    return s


def computeCompressedBits() -> int:
    s = 0
```

```
    for row in data:
        code = huffCode.get(makeToken(row))
        if code is not None:
            s += len(code)
    return s


def gcd(a: int, b: int) -> int:
    a = abs(a)
    b = abs(b)
    if b == 0:
        return 1 if a == 0 else a
    # Euclidean algorithm
    while b:
        a, b = b, a % b
    return a


# ==========================
# Utilities / UI
# ==========================


def safeInt(row: List[str], idx: int) -> int:
    if idx < 0 or idx >= len(row):
        return 0
```

```
    try:
        return int(row[idx].strip())
    except:
        return 0


def showData(limit: int):
    if not data:
        print("[No data]")
        return
    c = 0
    for r in data:
        print(r)
        c += 1
        if c >= limit:
            break
    print(f"Displayed {min(limit, len(data))} of {len(data)}")


def printDictionary():
    print(f"{'Token':20s} {'Freq':8s} {'Code':15s} {'Bits':5s}")
    print("-------------------------------------------------------------")
    for t, code in huffCode.items():
        f = freqMap.get(t, 0)
        print(f"{t:20s} {f:<8d} {code:15s} {len(code):<5d}")
```

```
def readFile(fname: str) -> str:
    sb: List[str] = []
    try:
        with open(fname, "r", encoding="utf-8") as br:
            for l in br:
                sb.append(l.strip())
    except Exception as e:
        print(f"Read error: {e}", file=sys.stderr)
    return "".join(sb)


# ========================
# Menu loop (mirrors Java)
# ========================


def menuLoop():
    global rbAge, rbAbs
    while True:
        print("\n--- Menu ---")
        print("1 Load CSV into LinkedList")
        print("2 Display Data")
        print("3 Generate Huffman Codes")
        print("4 Show Huffman Dictionary & stats")
```

```
print("5 Encode Data ")
print("6 Decode Encoded Text")
print("7 Build Red-Black Tree Index (age & absences) with logs")
print("8 Display RBT insertion logs and trees")
print("9 Exit")
choice = input("Choice: ").strip()
if not choice:
    continue
try:
    c = int(choice)
    if c == 1:
        fn = input("CSV filename (default student-data.csv): ").strip()
        if not fn:
            fn = "student-data.csv"
        loadCsvRecursive(fn)
    elif c == 2:
        showData(50)
    elif c == 3:
        if not data:
            print("Load CSV first.")
            continue
        buildFrequency()
        buildHuffmanTree()
```

```
        generateCodes()
        print(f"Huffman codes generated. Tokens: {len(huffCode)}")
    elif c == 4:
        if not huffCode:
            print("Generate codes first.")
            continue
        printDictionary()
        writeDictionaryCsv("dictionary.csv")
        orig = computeOriginalBits()
        comp = computeCompressedBits()
        print(f"Original Size (bits): {orig}")
        print(f"Compressed Size (bits): {comp}")
        g = gcd(orig, comp)
        # avoid divide-by-zero on empty comp
        ratio_left = orig // g if g else 0
        ratio_right = comp // g if g else 0
        dec = (float(orig) / comp) if comp else float('inf')
        print(f"Compression Ratio: {ratio_left}:{ratio_right}  (decimal: {dec:.2f})")
    elif c == 5:
        if not huffCode:
            print("Generate codes first.")
            continue
        encodeData()
```

```
    elif c == 6:
        if not huffCode:
            print("Generate codes first.")
            continue
        decodeData()
    elif c == 7:
        if not data:
            print("Load CSV first.")
            continue
        rbAge = RedBlackTree()
        rbAbs = RedBlackTree()
        rbAge.clearLog()
        rbAbs.clearLog()
        for row in data:
            a = safeInt(row, ageCol)
            abs_ = safeInt(row, absencesCol)
            rbAge.insert(a)
            rbAbs.insert(abs_)
        print("Built RBTs for age and absences. Logs recorded.")
    elif c == 8:
        print("\n--- Age RBT Logs ---")
        logsA = rbAge.getLog()
        if not logsA:
```

```
            print("[No logs - build RBT first]")
        else:
            for s in logsA:
                print(s)
                print()
        print("\nAge tree keys (in-order):", rbAge.inOrderKeys())
        rbAge.printTree("Age Final Red Black Tree Structure:")
        print("\n--- Absences RBT Logs ---")
        logsB = rbAbs.getLog()
        if not logsB:
            print("[No logs - build RBT first]")
        else:
            for s in logsB:
                print(s)
                print()
        print("\nAbsences tree keys (in-order):", rbAbs.inOrderKeys())
        rbAbs.printTree("Absences Final Red Black Tree Structure:")
    elif c == 9:
        print("Exit.")
        return
    else:
        print("Invalid choice.")
except ValueError:
```

```
            print("Enter a number.")
        except Exception as ex:
            print(f"Error: {ex}")
            import traceback
            traceback.print_exc()


if __name__ == "__main__":
    menuLoop()
```