# Operating Systems - Project Report

**Group Members:**
Mohammad Shahmeer Ul Haq – 22K-4643 (Group Leader)
Roohan Ahmed – 22K-4611
Zaid Vohra – 22K-4195

## Title: Parallel Programming Comparison of Sorting Algorithms using Pthreads vs. OpenMP vs. Serial

**Introduction:**
The Project uses 3 Sorting Algorithms (Bubble Sort, Merge Sort, and Quick Sort) using Pthreads, OpenMP, Serial to sort a large data set (numbers of students in our case) to show their performance.

**Background**: The aim was to compare the performance of sorting algorithms using parallel programming techniques Pthreads and OpenMP against the traditional serial approach. The focus will be on three sorting algorithms: Bubble Sort, Merge Sort, and Quick Sort

**Project Specification:**

1. Data Preparation:
   - Obtain and organize an array comprising 10,000 student marks.
2. Sorting Implementations:
   - Develop serial and parallel implementations of Quick Sort, Merge Sort, and Bubble Sort.
   - Utilize multi-threading and OpenMP for parallel implementations to optimize performance.
3. Graphical User Interface:
   - Enable users to trigger sorting operations through the GUI.
   - Present sorted data and performance metrics (e.g., execution time).
   - Enable pagination for navigating through sorted results.
   - Create interactive buttons for navigating between different sorting operations and result pages.
4. Performance Metrics:
   - Calculate and display the duration of each sorting operation.
5. Robustness and Usability:
   - Ensure the application is resilient to errors and user-friendly.

**Problem Analysis:**

- **Data Management:** Efficiently managing an array of 10,000 integers while ensuring thread safety during parallel processing.
- **Algorithm Optimization:** Some sorting algorithms, such as Bubble Sort, are inherently slow and may not experience significant benefits from parallelization.
- **Concurrency Control:** Ensuring that multi-threaded and OpenMP implementations correctly synchronize access to shared resources to prevent data corruption.
- **Memory Administration:** Properly managing dynamic memory to prevent leaks, especially within threaded environments.
- **User Interface:** Maintaining a responsive and informative GUI during intensive computational operations.

**Solution Design (Project Detail, Functionality and features):**
Our project features a main screen that lets you exit the program or start it, after that you're met with a fetch data button that fetches marks for you and then displays an unsorted data set. The next screen you come up to lets you choose a sorting algorithm and then the method you want to choose to run that algorithm. Each screen also has an X button that takes you to the home screen.

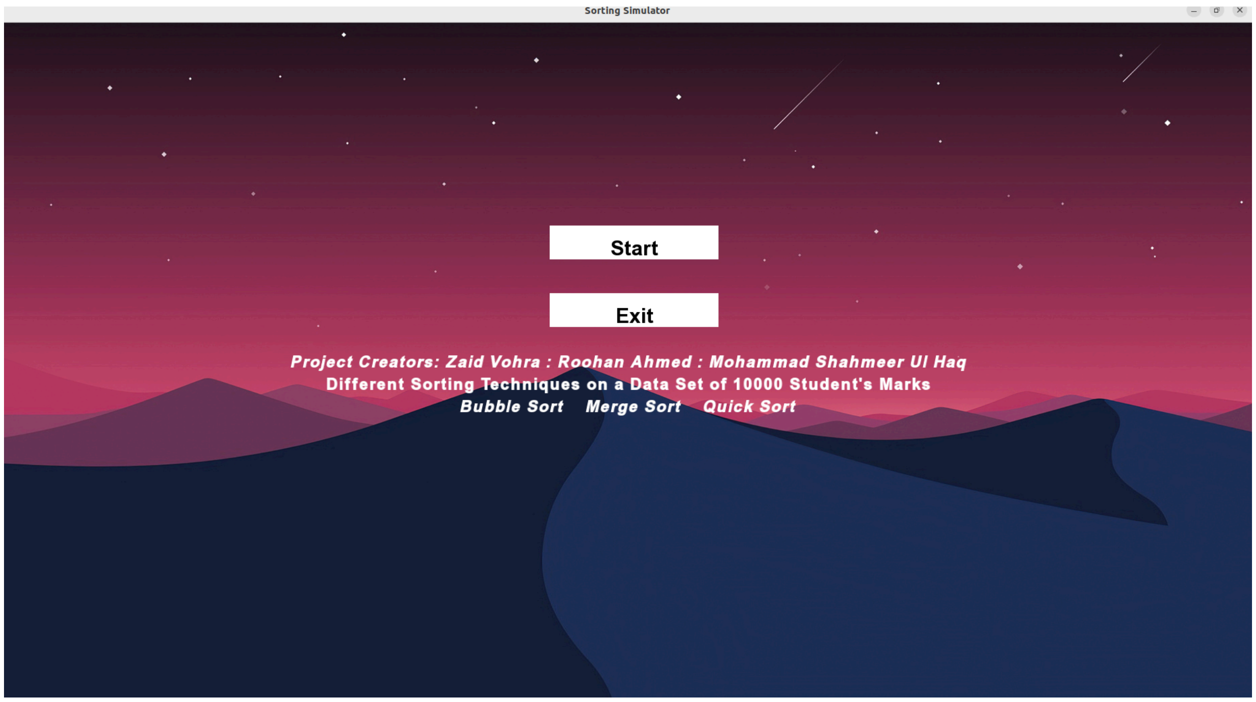
**Implementation & Testing:**

For the implementation phase, we divided the workload among group members to efficiently tackle different aspects of the project. Each member was responsible for implementing and testing specific sorting algorithms using different parallel programming techniques, we came up with many issues like the GUI not being compatible with printing large data sets or sorting algorithms mixing up data after being implemented with GUI.

**Project Breakdown Structure (Workload distribution with timeline):**
Mohammad Shahmeer Ul Haq: Wrote Codes for most of the sorting algorithms (threading and OpenMP of bubble sort, mergesort, Quicksort).
Zaid Vohra: Did majority of the GUI parts (from all the screens, menus, windows and button features while also helping out with serial parts of sorting codes).
Roohan Ahmed: Ran the GUI and sorting algorithms for error checking and fixed the issues that came along while helping out with problems during writing the codes, and helped format the gui properly.

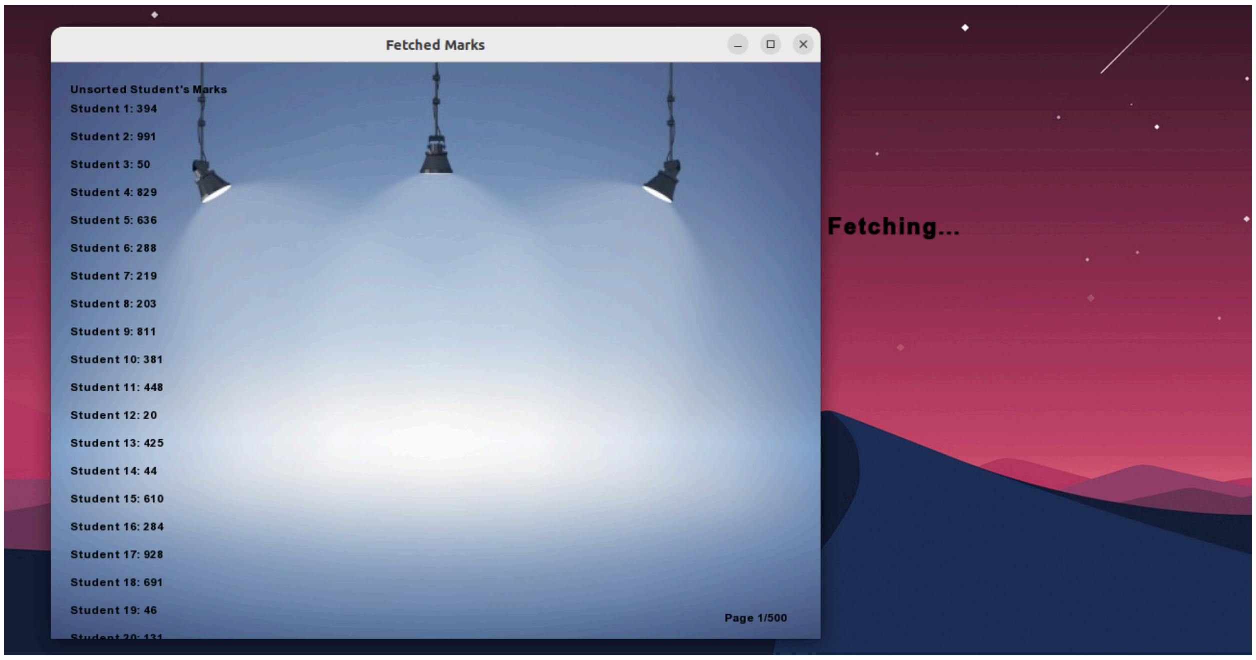
**Results (Outputs screenshots):**

Main Screen:



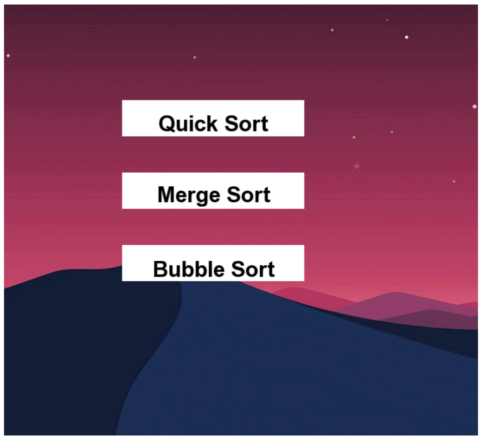
*Project Creators: Zaid Vohra : Roohan Ahmed : Mohammad Shahmeer Ul Haq*
Different Sorting Techniques on a Data Set of 10000 Student's Marks
*Bubble Sort* *Merge Sort* *Quick Sort*

Data Fetching:

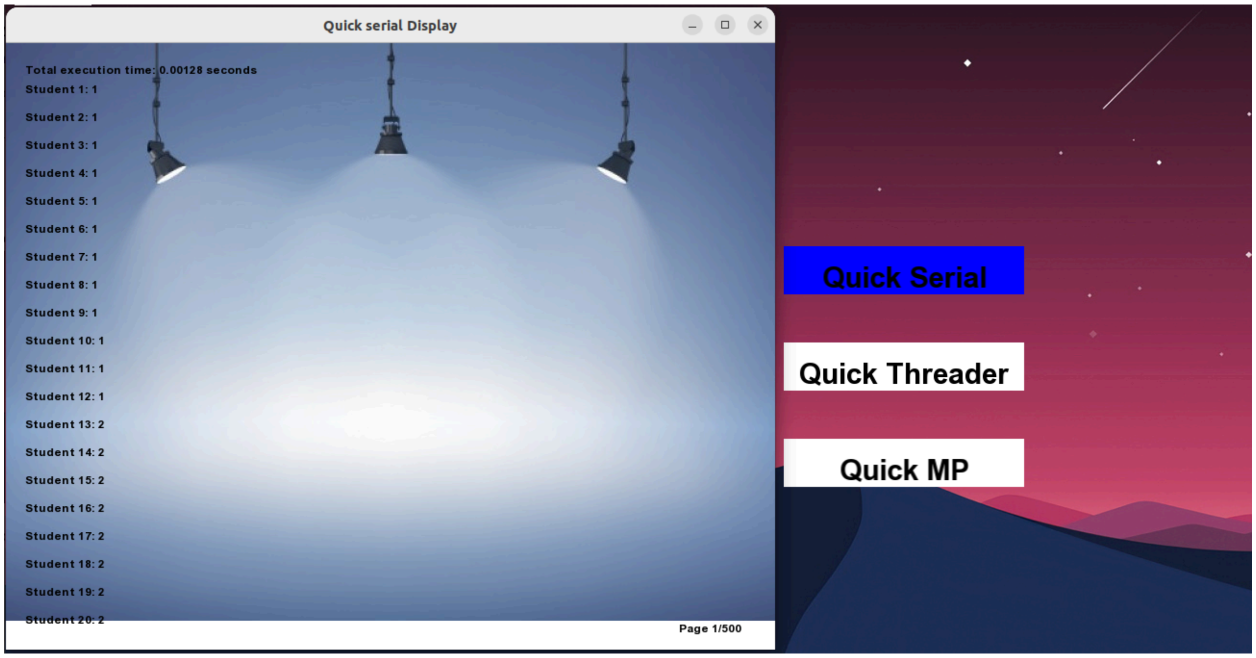


```
Unsorted Student's Marks
Student 1: 394
Student 2: 991
Student 3: 50
Student 4: 829
Student 5: 636
Student 6: 288
Student 7: 219
Student 8: 203
Student 9: 811
Student 10: 381
Student 11: 448
Student 12: 20
Student 13: 425
Student 14: 44
Student 15: 610
Student 16: 284
Student 17: 928
Student 18: 691
Student 19: 46
```

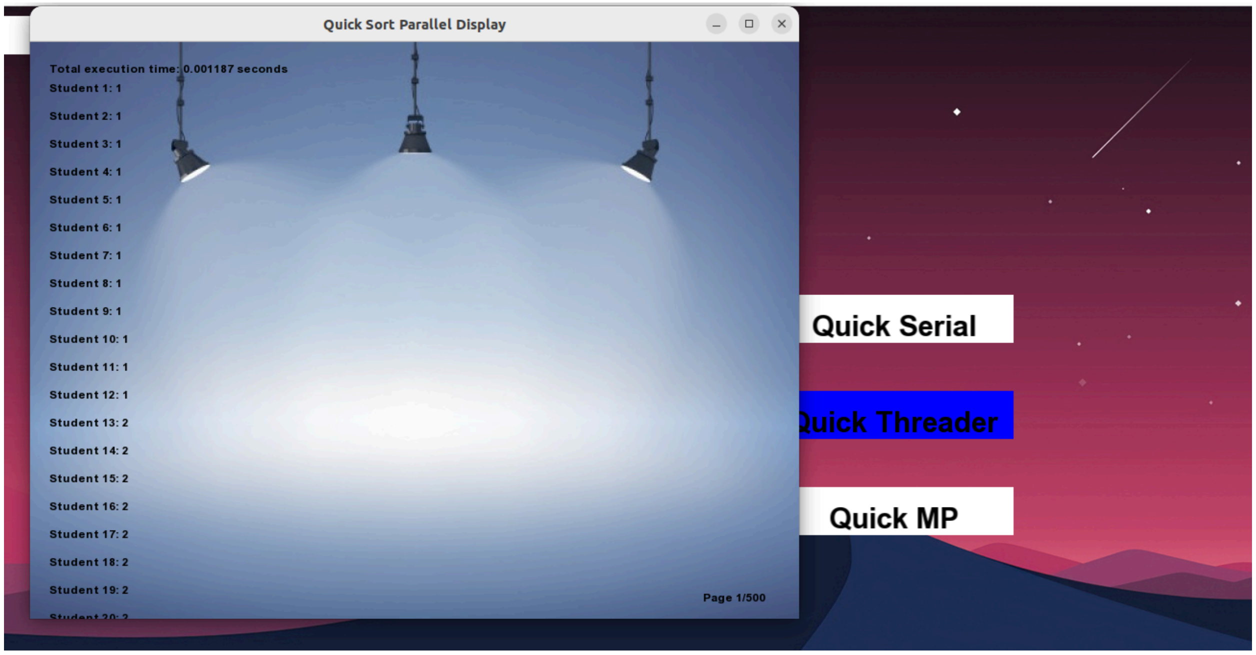
Sorting Algorithms after Fetching:



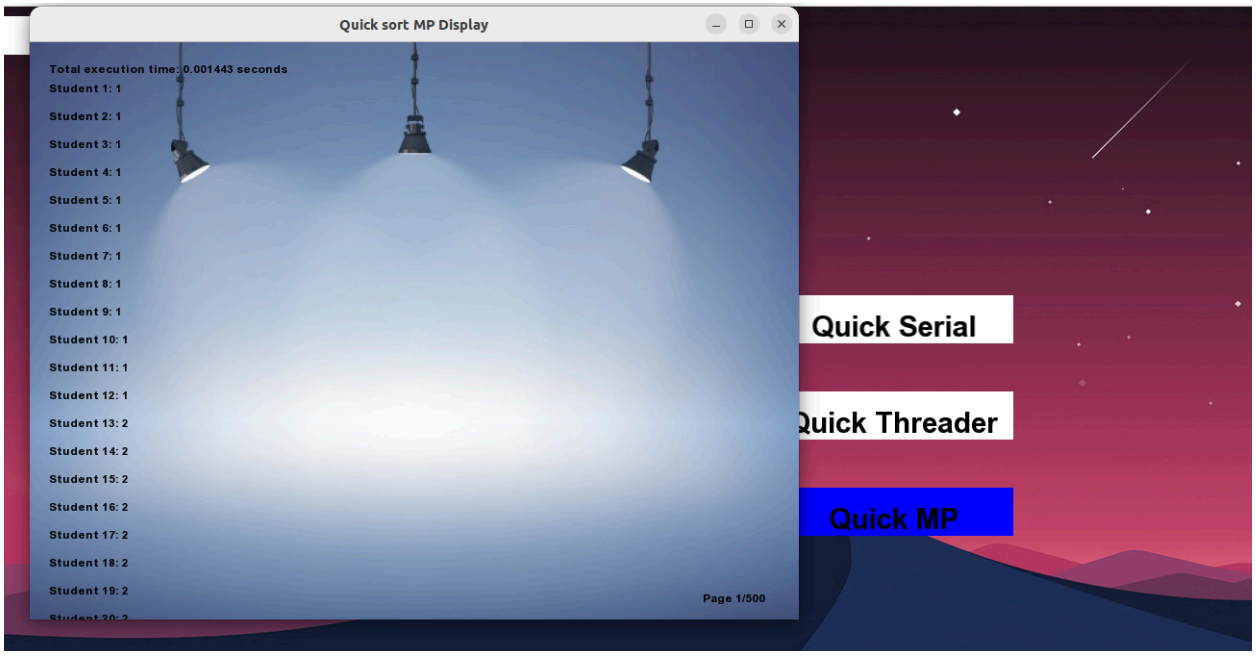
Quick Sort Outputs:



```
Total execution time: 0.00128 seconds
Student 1: 1
Student 2: 1
Student 3: 1
Student 4: 1
Student 5: 1
Student 6: 1
Student 7: 1
Student 8: 1
Student 9: 1
Student 10: 1
Student 11: 1
Student 12: 1
Student 13: 2
Student 14: 2
Student 15: 2
Student 16: 2
Student 17: 2
Student 18: 2
Student 19: 2
Student 20: 2
                                                  Page 1/500
```



```
Total execution time: 0.001187 seconds
Student 1: 1
Student 2: 1
Student 3: 1
Student 4: 1
Student 5: 1
Student 6: 1
Student 7: 1
Student 8: 1
Student 9: 1
Student 10: 1
Student 11: 1
Student 12: 1
Student 13: 2
Student 14: 2
Student 15: 2
Student 16: 2
Student 17: 2
Student 18: 2
Student 19: 2
Student 20: 2
Page 1/500
```



```
Total execution time: 0.001443 seconds
Student 1: 1
Student 2: 1
Student 3: 1
Student 4: 1
Student 5: 1
Student 6: 1
Student 7: 1
Student 8: 1
Student 9: 1
Student 10: 1
Student 11: 1
Student 12: 1
Student 13: 2
Student 14: 2
Student 15: 2
Student 16: 2
Student 17: 2
Student 18: 2
Student 19: 2
Student 20: 2
Page 1/500
```

## Conclusion (Summary & Discussion)

The project successfully integrates several sorting algorithms (Quick Sort, Merge Sort, Bubble Sort) applied to a dataset of 10,000 student marks, executed in both serial and parallel formats. The use of multi-threading, OpenMP, and an intuitive SFML-based GUI provides a multifaceted platform to visualize and analyze the performance of these algorithms under different conditions. This educational tool offers valuable insights into the complexities and efficiencies of various sorting methodologies, particularly illustrating the impact of parallel computing techniques on traditional algorithms.

- Algorithm Implementation and Comparison
- Graphical User Interface (GUI)
- Performance Analysis

The project successfully demonstrates the application of parallel processing techniques in improving the efficiency of sorting algorithms, an essential operation in many computing tasks. However, it also uncovers the limitations and considerations that must be addressed when choosing to parallelize certain operations:

- Scalability and Overhead
- Algorithm Suitability

In conclusion, this project serves as a robust platform for learning, analyzing, and visualizing the performance of sorting algorithms through both serial and parallel computing approaches, offering valuable insights into the practical applications and optimizations of data sorting.