Visualization of Page Replacement Algorithm: FIFO, LRU, and OPT Algorithms

Prepared By:

John Andrei Tacujan

BSCS – 3B

Submitted To:

Jo Anne G. Cura

Instructor

May 21, 2025

Table of Contents

Introduction

Imagine a computer system tasked with juggling multiple applications at once—web browsers, document editors, media players—all competing for limited memory space. As each application requests data, the operating system must make rapid decisions about which information to keep in memory and which to discard. These decisions are not random; they rely on carefully designed algorithms known as page replacement strategies, which aim to minimize delays caused by page faults and maximize system efficiency [1].

In operating system design, understanding these strategies is essential. However, understanding their complexities through textbook descriptions and static illustrations might be difficult. To bridge this gap, this project presents a **Page Replacement Algorithm Simulator**—a Python-based software application built using the Tkinter framework. The simulator brings the theory to life by offering a visual and interactive exploration of three foundational page replacement algorithms: **First-In-First-Out (FIFO), Least Recently Used (LRU), and Optimal (OPT).**

By enabling users to observe algorithm behavior with real-time visualization, the tool facilitates a deeper comprehension of how different strategies handle memory management under different scenarios. Research confirms that interactive visualization significantly enhances learning outcomes and engagement in computer science education [2]. This simulator serves as both an educational resource and a platform for performance comparison, making abstract concepts tangible and accessible.

Algorithm Descriptions

FIFO (First-In, First-Out)

The **First-In-First-Out (FIFO)** page replacement algorithm implements a straightforward approach to memory management based on temporal order of page loading. It uses queue-based mechanism ensures consistent and predictable behavior in page replacement decisions.

Least Recently Used (LRU) Algorithm

The **Least Recently Used (LRU)** algorithm implements a more sophisticated approach to page replacement by incorporating usage history into its decision-making process. This implementation maintains a dynamic record of page access patterns, consistently replacing the page that has remained unused for the longest period.

Optimal (OPT) Algorithm

The **Optimal page replacement algorithm** represents the theoretical best-case scenario for page replacement strategies. This implementation analyzes future page references to make optimal replacement decisions, effectively minimizing the total number of page faults that occur during program execution. While practically unimplementable in real systems due to its requirement for future knowledge, it serves as a valuable benchmark for evaluating other page replacement algorithms.

User Interface

The simulator's interface is divided into four main sections, each serving a specific purpose in the visualization and analysis of page replacement algorithms. The complete interface layout is shown in Figure 1.

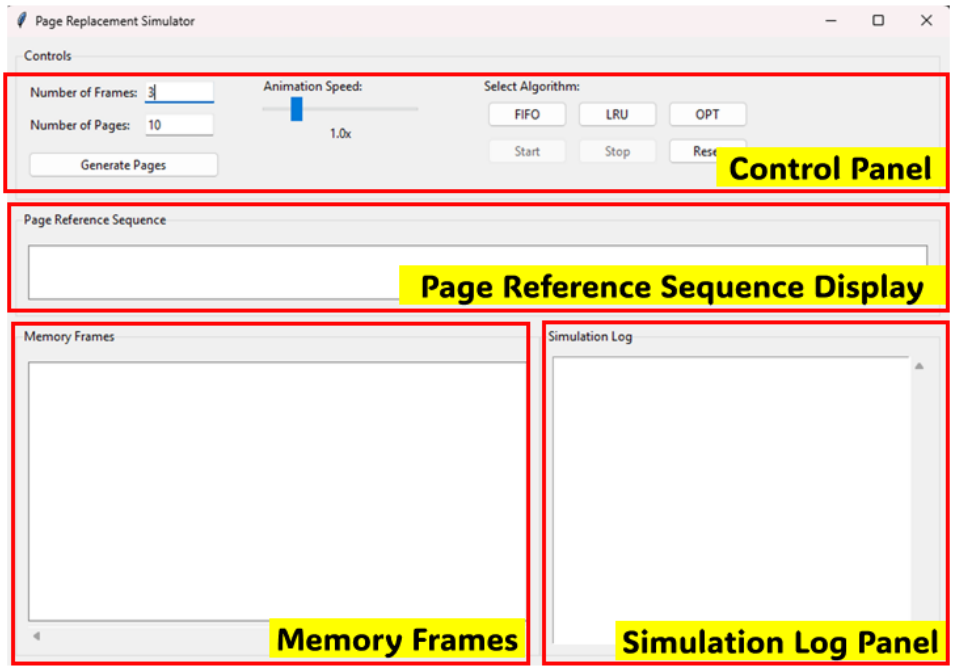

Figure 1: Main Interface

Controls Panel

The Controls Panel is the main user interface for configuring and controlling the simulation. It includes algorithm selection through three labeled radio buttons: FIFO, LRU, and OPT. The Configuration Controls section allows users to set the number of memory frames (ranging from 1 to 10, with a default of 3) and the length of the page reference sequence (1 to 50 pages, defaulting to 10). A "Generate Pages" button creates a new random sequence for testing. The Animation Controls enable speed adjustment (0.5x to 3.0x), simulation start/stop, reset to the initial state, and step-by-step execution for detailed analysis.

Page Reference Sequence Display

The Page Reference Sequence Display visually represents the simulated memory access pattern. The current page in focus is dynamically highlighted in red, allowing users to easily track simulation progress.

Memory Frames Visualization

The Memory Frames Visualization component forms the heart of the simulator, displaying the real-time state of memory frames through a grid-based layout. Each row represents a memory frame, while columns illustrate the temporal progression of memory states. This design creates a clear visual history of how pages move through memory over time.

Simulation Log Panel

The Simulation Log Panel is a tool for monitoring the algorithm's operation in real time. It combines textual logs with statistical insights to provide a complete record of the simulation's execution.

Sample Output 1 [4, 0, 7, 0, 9, 5, 7, 2, 6, 3].

To demonstrate the behavior and characteristics of each page replacement algorithm, we present sample executions with identical input parameters: 3 memory frames and a page reference sequence of [4, 0, 7, 0, 9, 5, 7, 2, 6, 3].

FIFO Algorithm Sample Output

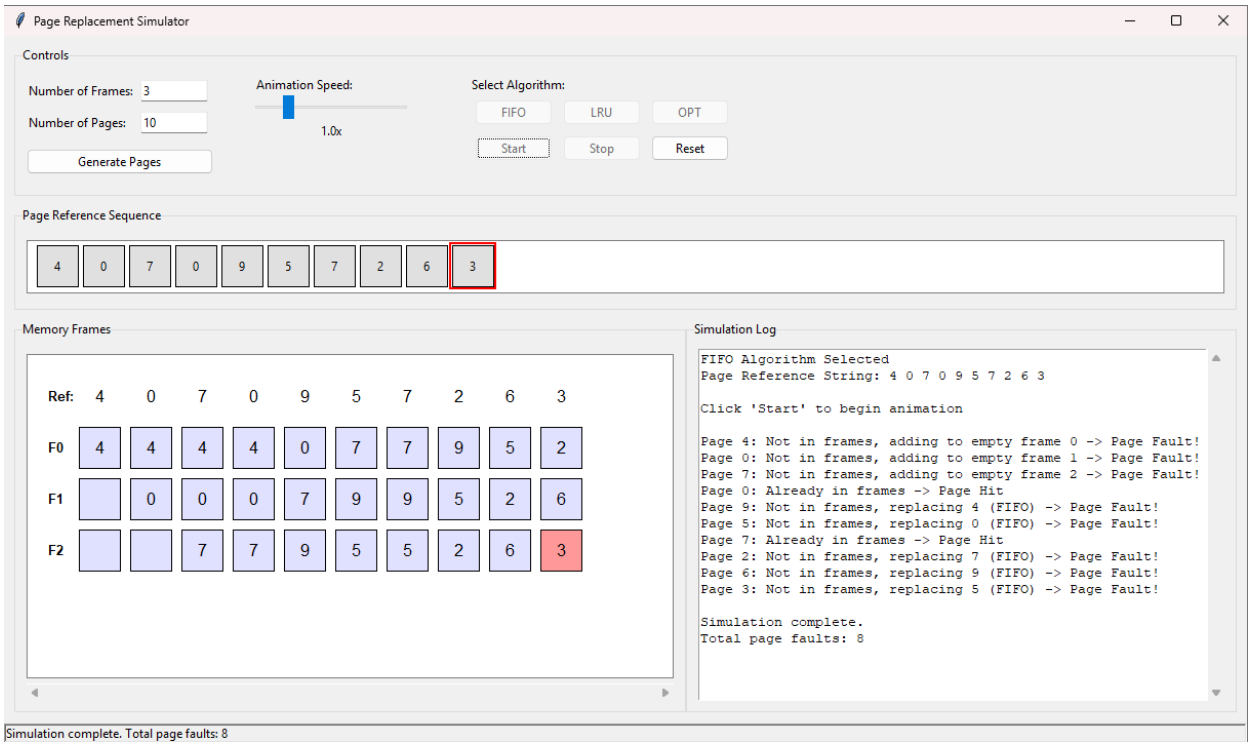

Figure 2: Sample Output 1: FIFO Algorithm

This demonstrates the FIFO algorithm's execution. The visualization shows how pages are replaced in the exact order they were loaded, regardless of their usage frequency. Using three frames, FIFO fills empty frames with pages 4, 0, and 7. The first page hit occurs when page 0 reappears. Pages 9, 5, 2, 6, and 3 each cause faults by replacing the oldest pages (4, 0, 9, 5, and 7 respectively). A second hit occurs with page 7's reappearance. Final state: [2, 6, 3], with 8-page faults (80% miss rate), showing FIFO's weakness in handling temporal locality.

LRU Algorithm Sample Output

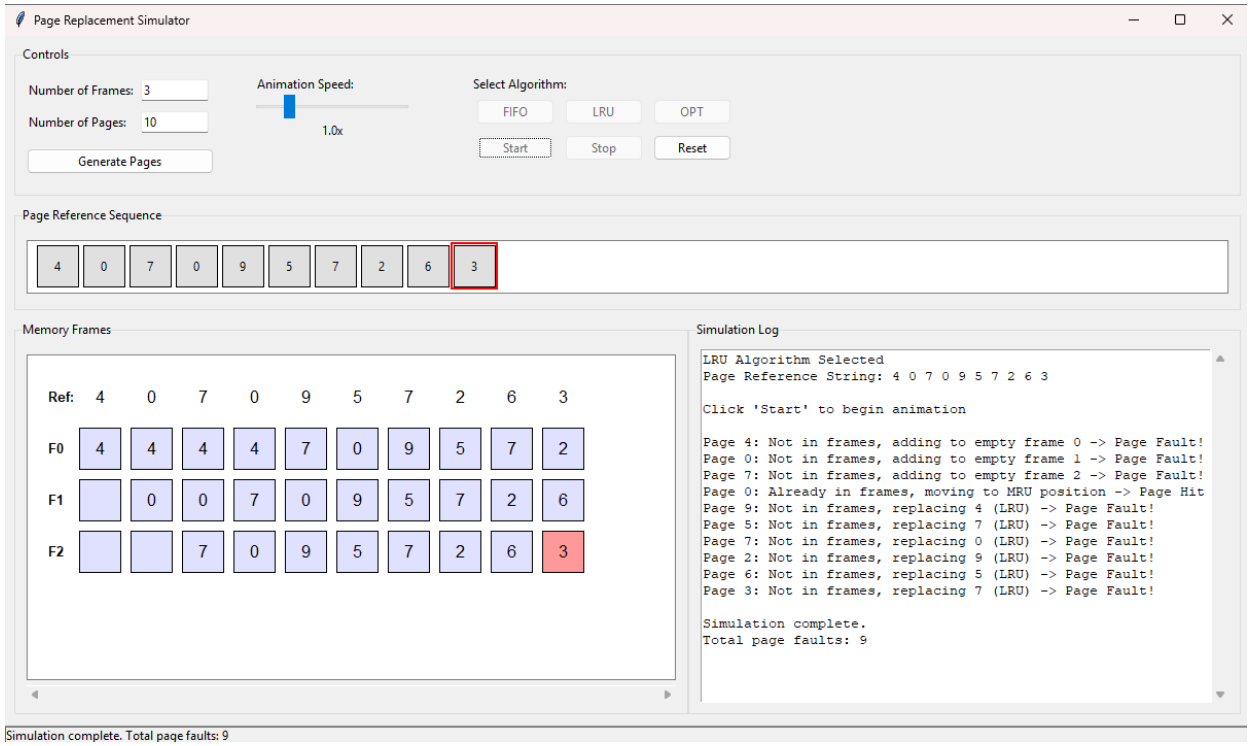

Figure 3: Sample Output 1: LRU Algorithm

This shows LRU processing the same sequence. The algorithm begins by filling empty frames with pages 4, 0, and 7. When page 0 appears again, it's a hit and moves to the most recently used position. Page 9 replaces 4 (least recently used), followed by 5 replacing 7, then 7 returns replacing 0. Pages 2, 6, and 3 subsequently replace 9, 5, and 7 respectively. Final state: [2, 6, 3], with 9-page faults (90% miss rate), showing how LRU's tracking of usage patterns can still result in high fault rates with certain reference patterns.

OPT Algorithm Sample Output

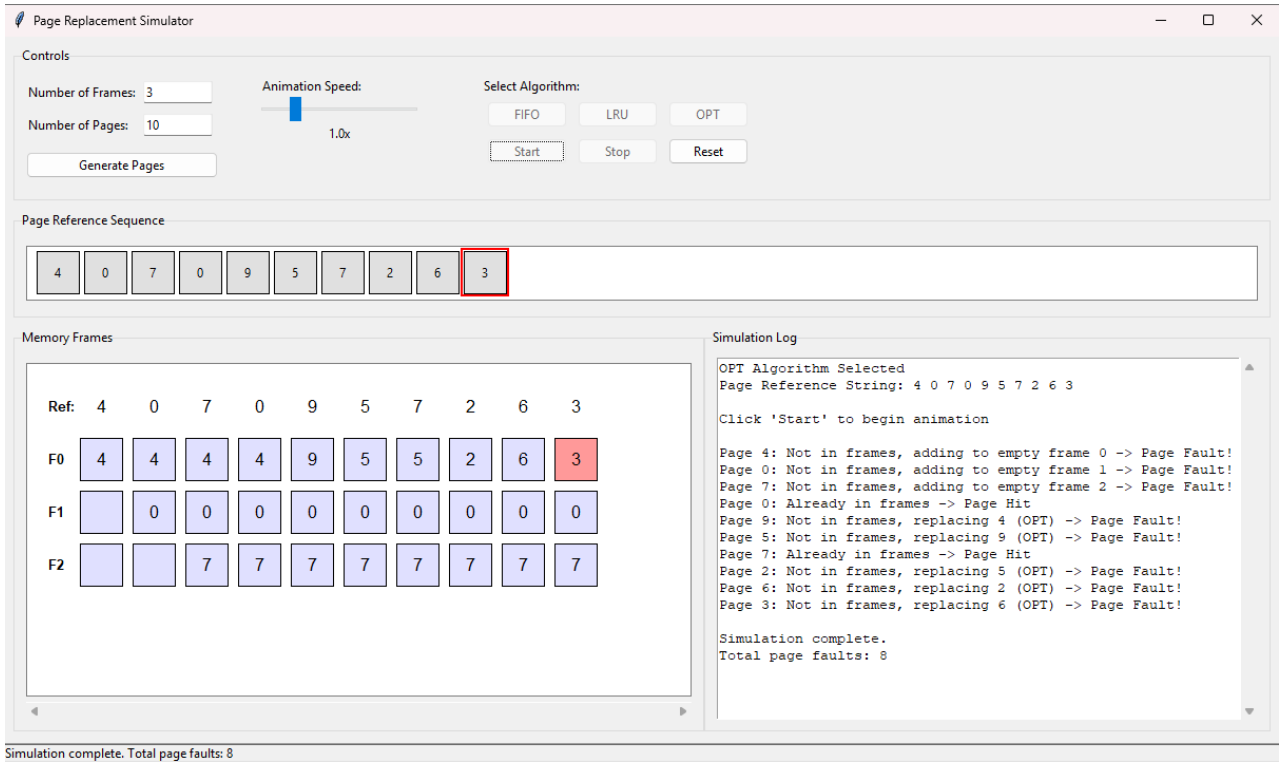

Figure 4: Sample Output 1: OPT Algorithm

Figure 5 illustrates OPT's execution with reference sequence [4, 0, 7, 0, 9, 5, 7, 2, 6, 3]. The algorithm begins by filling empty frames with pages 4, 0, and 7. A page hit occurs with page 0's second reference. Using future knowledge, it replaces page 4 with 9, then 9 with 5 (as 9 won't be used again). After a second hit with page 7, it makes optimal replacements for pages 2, 6, and 3. Final state: [7, 0, 3], with 8-page faults (80% miss rate), demonstrating the minimum possible faults for this sequence.

Sample Output 2 [6, 8, 8, 7, 5, 7, 5, 9, 8, 1, 3, 2, 8, 7, 8]

To demonstrate the behavior and characteristics of each page replacement algorithm, we present sample executions with identical input parameters: 3 memory frames and a page reference sequence of [6, 8, 8, 7, 5, 7, 5, 9, 8, 1, 3, 2, 8, 7, 8].

FIFO Algorithm Sample Output

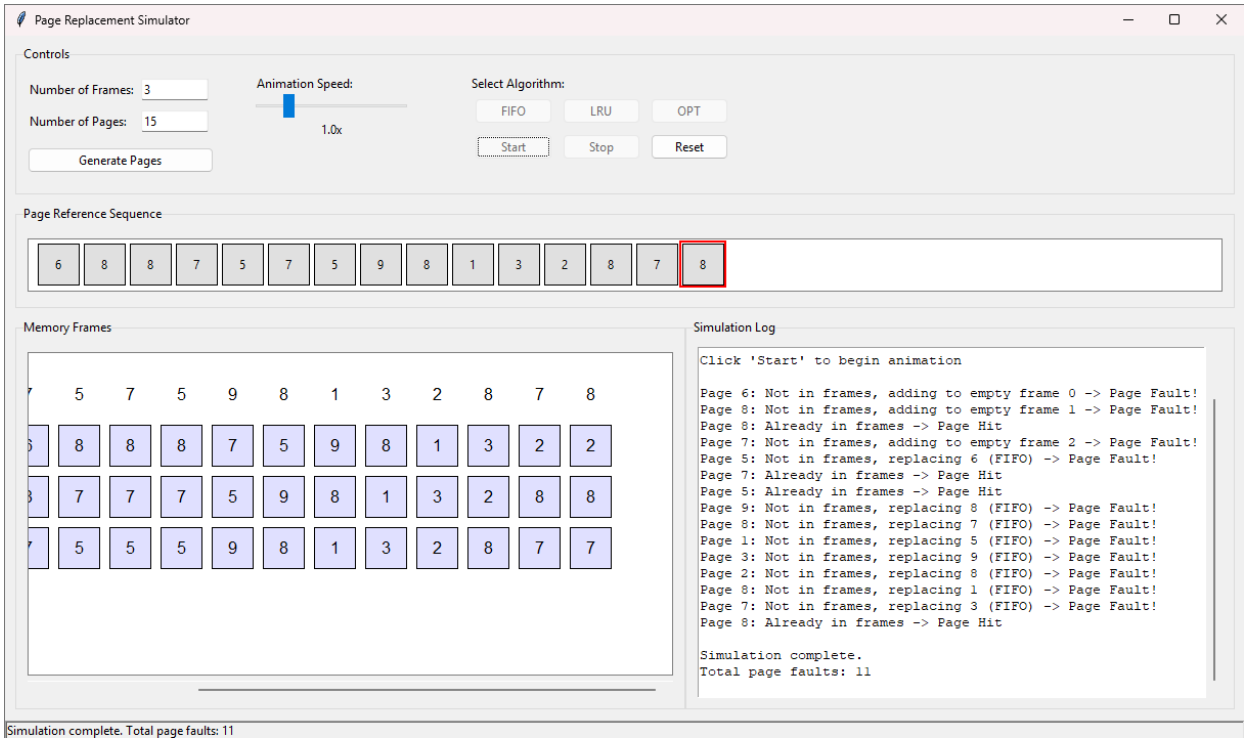

Figure 5: Sample Output 2: FIFO Algorithm

This demonstrates the FIFO algorithm's execution. The visualization shows how pages are replaced in the exact order they were loaded, regardless of their usage frequency. Using three frames, FIFO fills empty frames with pages 6, 8, and 7. The first page hit occurs when page 8 appears second time. Pages 5, 7, and 5 create a sequence where 5 replaces 6, 7 replaces 8, and 5 remains (hit). When page 9 arrives, it replaces 7. Subsequent pages 8, 1, 3, 2 replace the oldest pages in memory following the FIFO principle. The sequence ends with hits from pages 8 and 7. Final state: [2, 8, 7], with 11 page faults (73.3% miss rate), demonstrating FIFO's inability to consider page usage frequency.

LRU Algorithm Sample Output

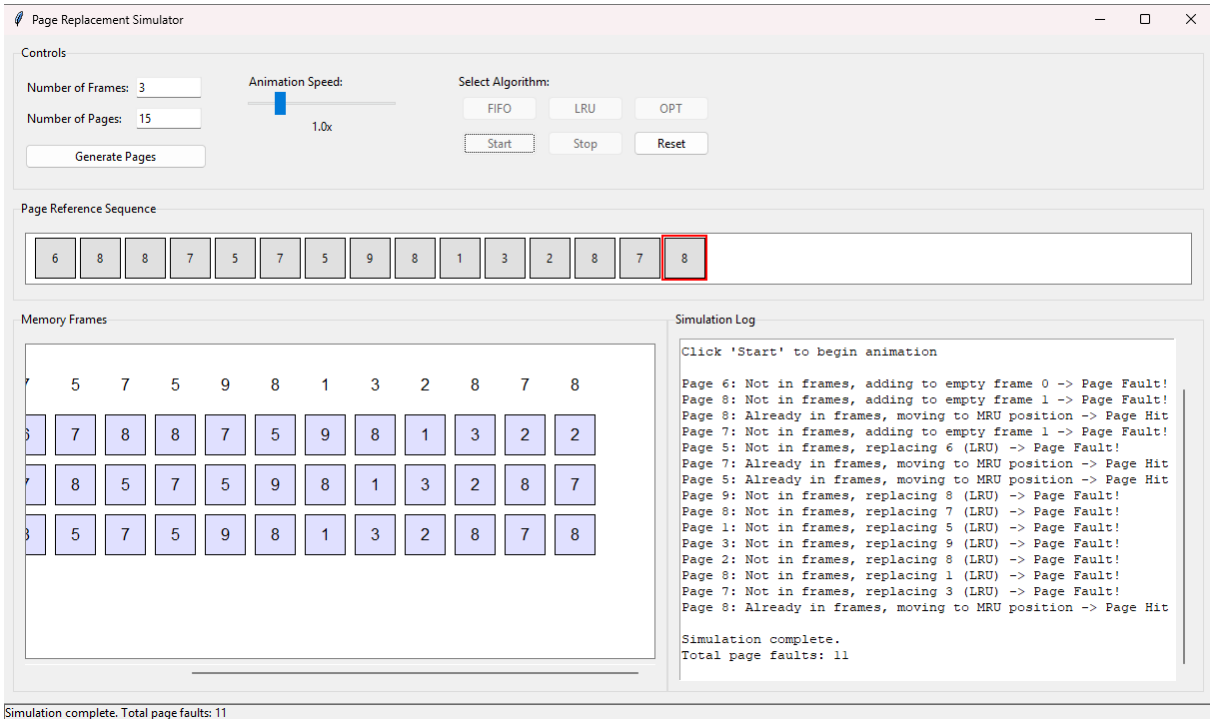

Figure 6: Sample Output 2: LRU Algorithm

This shows LRU processing the same sequence. The algorithm begins by filling empty frames with pages 6 and 8. A hit occurs when page 8 appears again. Page 7 is then added to an empty frame. Page 5 replaces 6 (least recently used), followed by a hit when 7 appears again. The next 5 creates a hit, moving it to the most recently used position. Page 9 replaces 8 (LRU), then 8 returns replacing 7. Pages 1, 3, and 2 cause faults by replacing the least recently used pages in sequence. Page 8 replaces 1, followed by 7 replacing 3, and finally a hit when 8 appears again. Final state: [2, 8, 7], with 11-page faults (73.3% miss rate), showing equal performance to FIFO in this sequence.

OPT Algorithm Sample Output

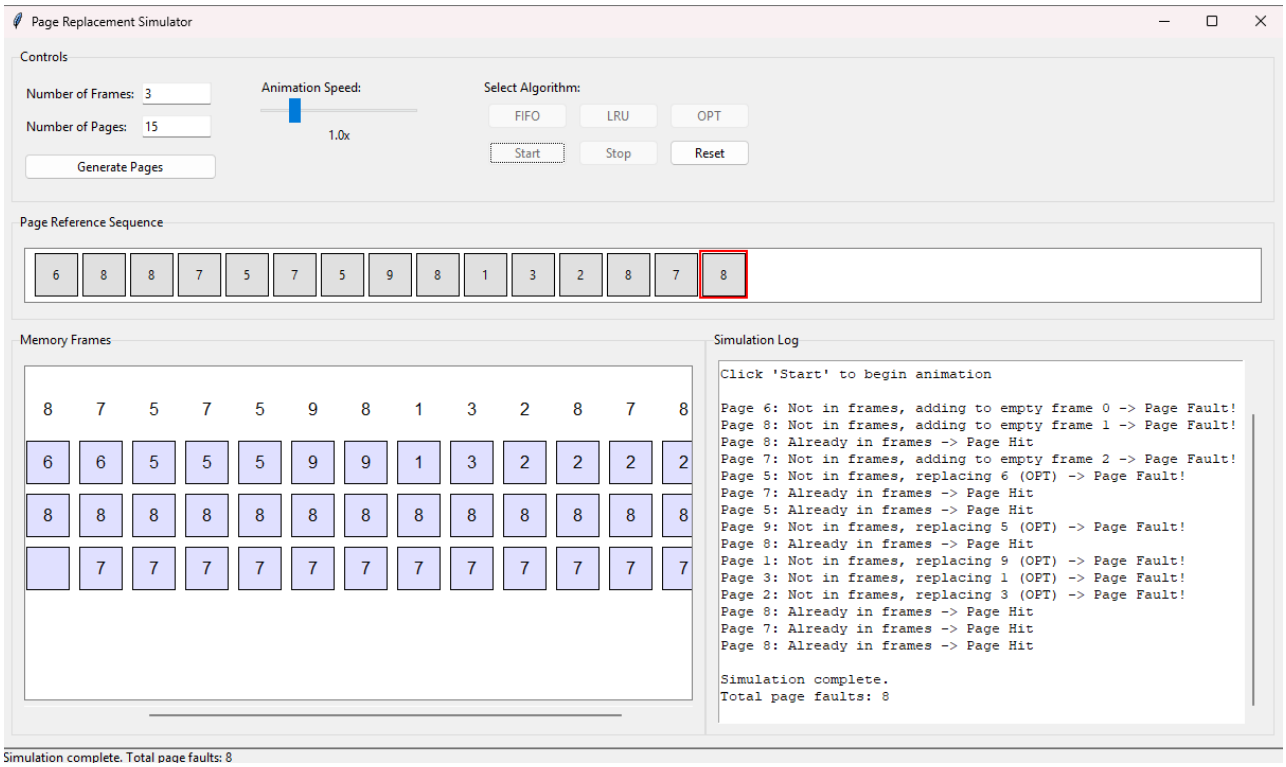

Figure 7: Sample Output 2: OPT Algorithm

This illustrates OPT's execution with the same reference sequence. The algorithm starts by filling empty frames with pages 6, 8, and 7. The second reference to page 8 creates a hit. Using its future knowledge, OPT makes optimal replacement decisions: page 5 replaces 6 (as 6 won't be used again), and subsequent replacements are made by looking ahead at future references. The algorithm maintains pages that will be needed soonest, resulting in fewer page faults. Final state: [2, 8, 7], with 9 page faults (60% miss rate), demonstrating the optimal performance possible for this sequence.

Sample Output 3 [9, 2, 7, 9, 6, 0, 7, 7, 7, 5, 4, 6, 4, 5, 6]

To demonstrate how the algorithms behave with a different frame count, we analyze their execution with 4 memory frames and a page reference sequence of [9, 2, 7, 9, 6, 0, 7, 7, 7, 5, 4, 6, 4, 5, 6]. This sequence features multiple consecutive references to the same page (e.g., three 7's in a row) and repeated patterns, providing interesting insights into how each algorithm handles different access patterns.

FIFO Algorithm Sample Output

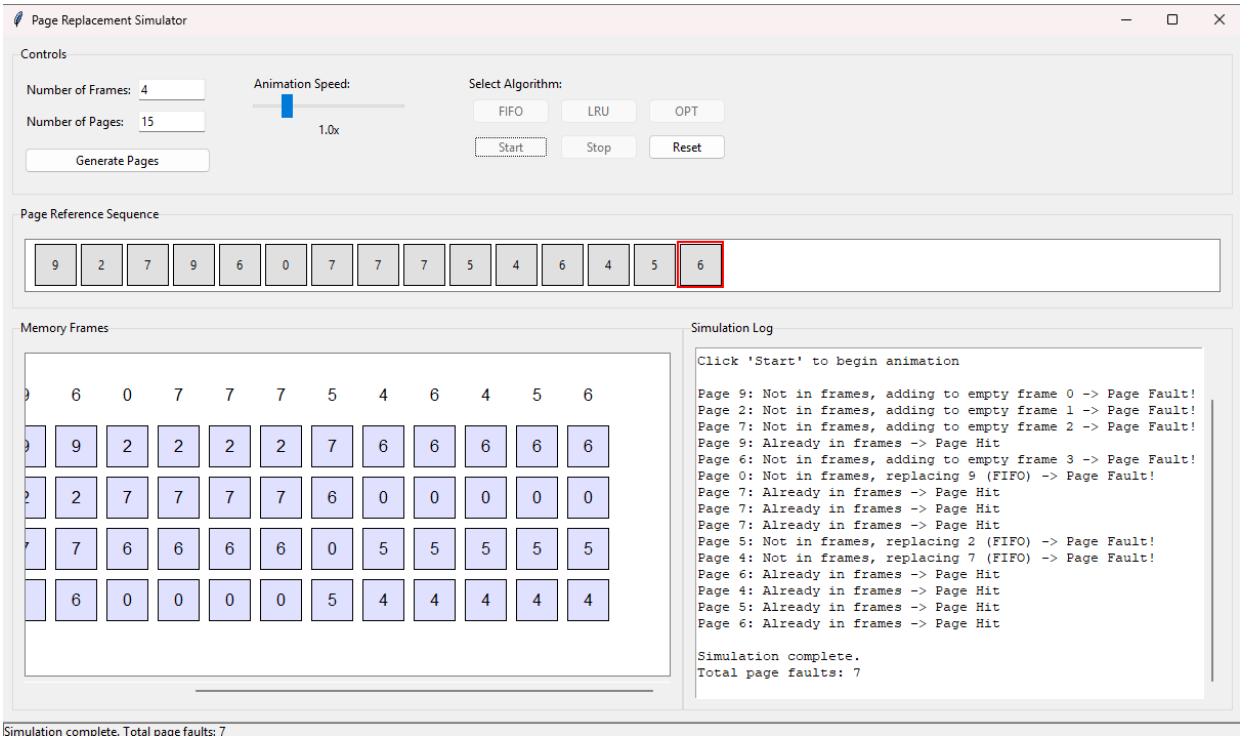

Figure 8: Sample Output 3: FIFO Algorithm with 4 Frames

The First-In-First-Out algorithm demonstrates efficient handling of this sequence with four frames available. Initially, the algorithm fills three frames with pages 9, 2, and 7, experiencing faults for each. When page 9 appears again, it results in the first hit as the page is still in memory. The algorithm then adds page 6 to the last empty frame. As new pages arrive, FIFO follows its replacement policy strictly: page 0 replaces 9 (the oldest), then encounters three consecutive hits with page 7, showcasing how FIFO can benefit from temporal locality despite not explicitly tracking it. Page 5 causes a fault and replaces 2 (now the oldest), followed by page 4 replacing 7. The sequence concludes with four consecutive hits on pages 6, 4, 5, and 6, as all needed pages remain in memory. The final state shows [6, 0, 5, 4], with only 7-page faults (46.7% miss rate), demonstrating how FIFO can perform well with adequate frame count and favorable reference patterns.

LRU Algorithm Sample Output

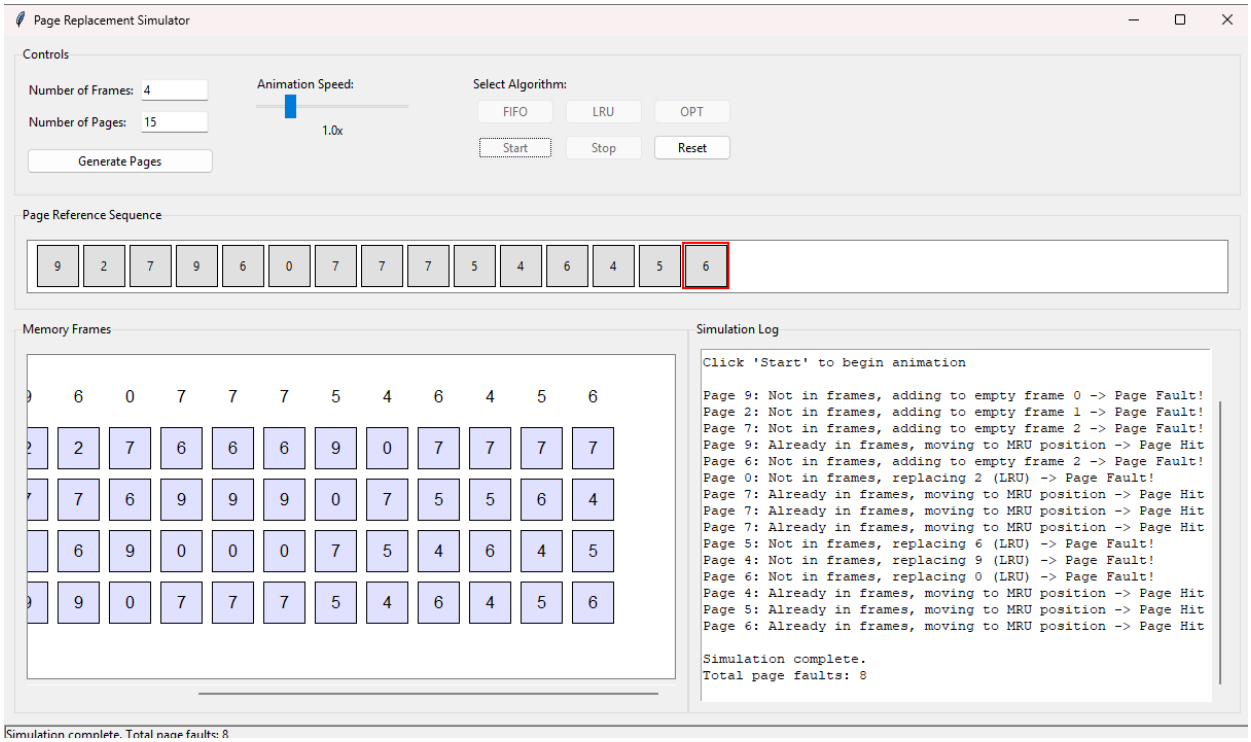

Figure 9: Sample Output 3: LRU Algorithm with 4 Frames

The Least Recently Used algorithm processes this sequence while maintaining usage recency information. The algorithm begins similarly by loading pages 9, 2, and 7, but when page 9 appears again, LRU updates its position to most recently used. Page 6 fills the final frame, then page 0 causes a fault replacing 2 (the least recently used). The consecutive references to page 7 demonstrate LRU's handling of repeated accesses - each hit moves 7 to the most recently used position. When page 5 arrives, it replaces page 9, which hadn't been used since its early reappearance. The sequence continues with page 4 replacing 6, then 6 returns replacing 0. The final series sees 4, 5, and 6 each being moved to most recently used position upon access. The algorithm concludes with state [7, 4, 5, 6], accumulating 8-page faults (53.3% miss rate). Surprisingly, LRU performs slightly worse than FIFO in this case, illustrating how recency-based decisions aren't always optimal.

OPT Algorithm Sample Output

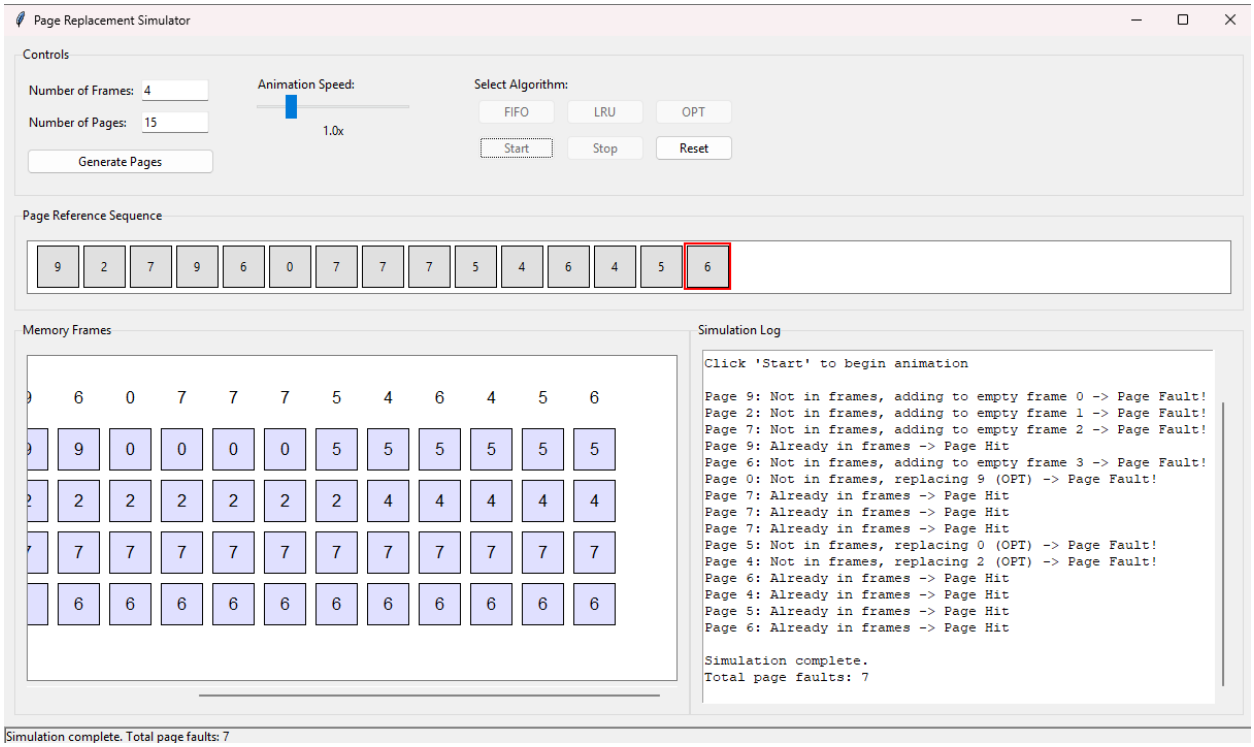

Figure 10: Sample Output 3: OPT Algorithm with 4 Frames

The Optimal algorithm demonstrates its prescient decision-making capabilities with this sequence. Like the others, it begins by loading pages 9, 2, and 7 into empty frames, experiencing faults for each. When page 9 appears again, it results in a hit. Page 6 then fills the final empty frame. The algorithm makes its first optimal choice by replacing page 9 with 0, recognizing that among the current pages in memory, 9 is the one that won't be needed for the longest time. The subsequent triple occurrence of page 7 results in three consecutive hits, showcasing efficient frame utilization. When page 5 arrives, OPT replaces page 0, followed by page 4 replacing page 2, as these choices optimize future access patterns. The sequence concludes with five consecutive hits as pages 6, 4, 5, and 6 are accessed, with all needed pages already in memory. The final state [5, 4,7, 6] is achieved with 7-page faults (46.7% miss rate), demonstrating OPT's ability to minimize page faults through its future knowledge.

Conclusion

This project has successfully demonstrated the value of interactive visualization in understanding page replacement algorithms while providing a framework for algorithm analysis and comparison. Through the implementation of FIFO, LRU, and OPT algorithms in a simulated environment, the tool has highlighted the behavioral differences and performance characteristics of each strategy under varying memory access patterns.

The insights gained contribute to the educational area of operating system memory management. The consistent superiority of the Optimal algorithm in minimizing page faults affirms its theoretical advantage [1], while the behavior of FIFO and LRU under different scenarios reflects their practical strengths and limitations. These findings reinforce the importance of selecting memory management strategies based on workload characteristics.

The simulator demonstrates the effectiveness of visual learning tools in improving comprehension of abstract concepts. Research suggests that interactive visualization significantly improves understanding and retention of algorithmic processes in computing education [2]. By enabling users to observe and interact with the algorithmic processes in real time, the project supports active learning and deepens conceptual understanding—an approach aligned with modern pedagogical practices in computer science education.

References

[1] ABRAHAM SILBERSCHATZ, PETER BAER GALVIN, GREG GAGNE, Operating System Concept Tenth Edition, Laurie Rosatone, 2018.

[2] Slavomir Simonak, "Using algorithm visualizations in computer science education," Open Computer Science, October 2014. [Online]. Available: https://www.researchgate.net/publication/273496790_Using_algorithm_visualizations_in_computer_science_education.