RV College of Engineering®

Go, change the world

DEPARTMENT OF INFORMATION SCIENCE AND ENGINEERING

OPERATING SYSTEM LAB EL REPORT

CS235AI

Submitted by

Rohan R Gowda **1RV23IS099**

Page Replacement Algorithm

Under the guidance of

Sushmitha N
Assistant Prof.
Dept. of ISE
RV College of Engineering®

2024-2025

Table of Contents

Introduction

In modern computing systems, memory management plays a critical role in ensuring the efficient performance of applications and overall system functionality. One key component of memory management is page replacement algorithms, which are designed to manage the limited amount of physical memory available for running processes. When a process needs more memory than what is physically available, the operating system must decide which pages of memory should be swapped out to disk to make space for the new data. The challenge lies in choosing the most appropriate page to evict, ensuring that the system operates efficiently with minimal performance degradation.

This report explores various page replacement algorithms, focusing on their principles, implementation, and performance in different scenarios. The algorithms we will discuss include the First-In-First-Out (FIFO), Least Recently Used (LRU), and Optimal Page Replacement, each with its unique approach to managing memory.

To make these concepts more accessible and easier to understand, this project incorporates an interactive web-based simulation created with HTML and CSS. This simulation allows users to visualize how different page replacement strategies work in real-time, offering a practical demonstration of the theoretical concepts.

By examining these algorithms in detail, we aim to highlight their strengths, weaknesses, and suitability for different system environments. Ultimately, this report will provide insights into how efficient page replacement strategies can optimize memory utilization, improve system performance, and support the overall reliability of computing systems.

Problem Statement

With the increasing complexity and demands of modern applications, efficient memory management has become a crucial aspect of operating systems. One of the key challenges in memory management is **page replacement**, which deals with selecting which pages to swap out when the physical memory is full. Inefficient page replacement can lead to issues such as excessive disk I/O, increased latency, and degraded system performance, commonly known as **thrashing**.

The problem this report addresses is how to design and evaluate different **page replacement algorithms** that determine which page in memory should be replaced when a page fault occurs. Specifically, the goal is to explore how different algorithms (such as **FIFO**, **LRU**, and **Optimal Page Replacement**) handle page faults and manage the limited memory resources effectively.

This project aims to provide a practical understanding of page replacement strategies by not only explaining the theoretical aspects but also by offering a real-time simulation to visualize and analyze their behavior. Ultimately, this will help in identifying the algorithm best suited for different memory management scenarios.

Objectives

1. To Simulate Page Replacement Algorithms:

   To create an interactive simulation of First-In-First-Out (FIFO), Least Recently Used (LRU), and Optimal Page Replacement algorithms. The simulation will allow users to input page reference strings and memory sizes to observe how each algorithm handles page faults.

2. To Track Hit and Fault Rates:

   To calculate and display the hit rate (percentage of times the page is found in memory) and fault rate (percentage of times the page needs to be loaded into memory) for each page replacement algorithm during the simulation.

3. To Count Number of Hits and Faults:

   To count and show the total number of page hits and page faults during the simulation. This data will help users understand the efficiency of each algorithm based on their memory access patterns.

4. To Provide Real-Time Feedback:

   To provide real-time feedback during the simulation, showing how pages are replaced in memory and how the algorithms respond to page faults, offering a dynamic visualization of memory management.

5. To Compare Algorithm Performance:

   To compare the performance of the different algorithms in terms of hit/fault rates and number of faults/hits under various scenarios, such as varying memory sizes and access patterns. This will help in understanding the strengths and weaknesses of each algorithm.

6. To Enhance Understanding Through Visualization:

   To provide a clear and interactive visual representation of each algorithm's operation, making it easier for users to understand the mechanics of page replacement and how different algorithms perform in managing memory.

OS concepts used

This project involves several key Operating System (OS) concepts, particularly related to memory management and page replacement algorithms. Below are the main OS concepts utilized in this project:

1. Memory Management:

   Memory management is a fundamental concept in operating systems, responsible for coordinating and controlling the computer’s memory resources. It ensures that processes have enough memory to execute while preventing one process from interfering with the memory allocated to another. This project specifically focuses on virtual memory management, where the operating system manages a combination of physical memory and secondary storage (disk space).

2. Page Faults:

   A page fault occurs when a process tries to access a page that is not currently loaded in the physical memory (RAM). When a page fault happens, the operating system must decide which page to replace in memory to bring the requested page from secondary storage (disk) into RAM. Handling page faults efficiently is a core aspect of memory management, and the simulation in this project demonstrates how page replacement algorithms manage these faults.

3. Page Replacement:

   When memory is full, and a new page needs to be loaded, the operating system must decide which page in memory to evict. This is done using page replacement algorithms, which determine the optimal strategy for replacing pages to minimize page faults. The algorithms simulated in this project are:

   - First-In-First-Out (FIFO): This algorithm replaces the oldest page in memory.
   - Least Recently Used (LRU): This algorithm replaces the page that has not been used for the longest period.
   - Optimal Page Replacement: This theoretical algorithm replaces the page that will not be used for the longest time in the future.

4. Virtual Memory:

   Virtual memory is a concept that allows a system to use more memory than is physically available by using disk storage as an extension of RAM. When a page is not in physical memory, it is swapped in from the disk, which involves page faults and triggers page replacement algorithms. This project simulates how virtual memory works by handling page faults and managing memory pages through algorithms.

5. Memory Frames:

   In the context of this project, memory frames refer to fixed-size blocks of physical memory (RAM) where pages are stored. The number of memory frames determines the total amount of memory available for page storage. The project allows users to modify the number of memory frames to see how different page replacement algorithms perform under varying memory sizes.

6. Page Table:

   Although not explicitly simulated in this project, the page table is an essential OS concept that maps virtual addresses (used by programs) to physical addresses (in RAM). The page replacement algorithms in this project mimic this behavior by deciding which virtual pages should be loaded into physical memory.

7. Thrashing:

   Thrashing is a situation where the operating system spends more time swapping pages in and out of memory than executing processes. This happens when there are too many page faults, and the system is overwhelmed by the need to constantly replace pages. By comparing the performance of different page replacement algorithms, this project indirectly highlights how algorithms can help avoid or mitigate thrashing by reducing page fault rates.

By understanding these core OS concepts, the project provides insights into how operating systems manage memory, handle page faults, and optimize system performance using different page replacement strategies.

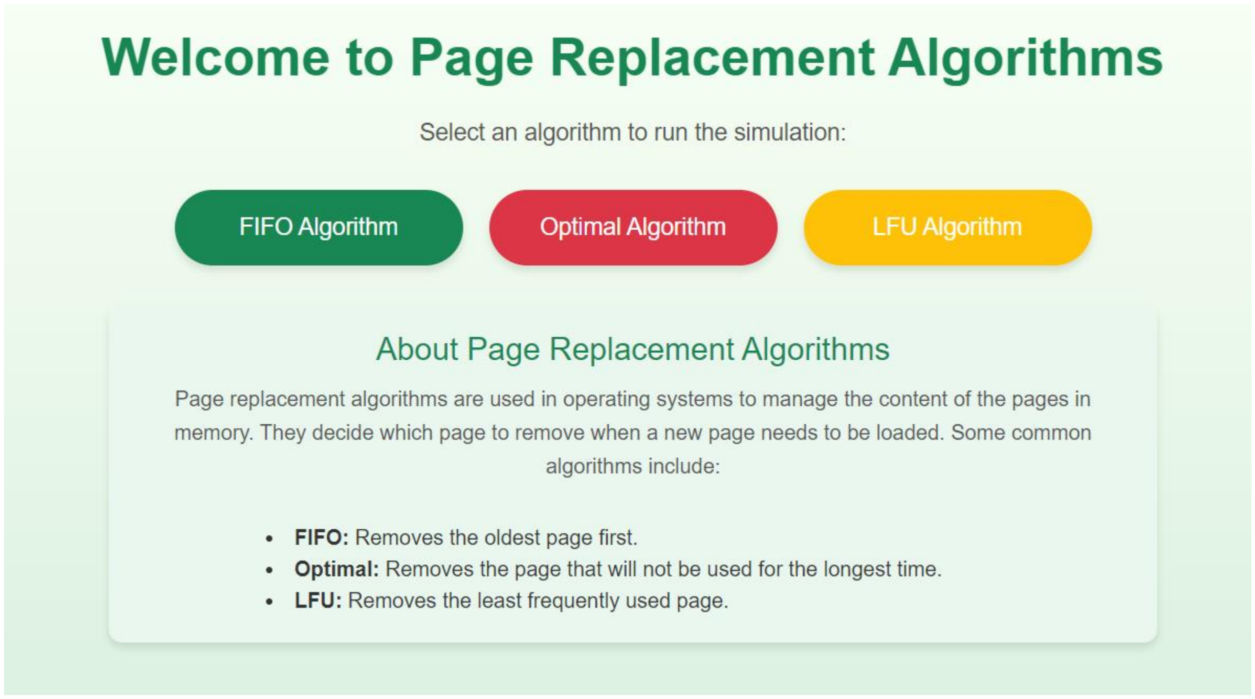
Implementation details

This project uses HTML, CSS, JavaScript **(JS)** and Bootstrap to create a visually appealing and interactive interface for simulating different Page Replacement Algorithms (FIFO, Optimal, and LFU). JavaScript is included in this specific implementation for dynamic functionality, the page layout and structure focus on providing a clean, responsive interface for easy navigation through HTML and CSS.

Technologies Used:

1. HTML (Hypertext Markup Language):
    - HTML is used to create the basic structure of the webpage. It organizes the content, including the title, headings, buttons, and links for navigating to different page replacement algorithm simulations.
2. CSS (Cascading Style Sheets):
    - CSS is used for styling the webpage and ensuring a modern and visually attractive layout. It customizes the design of buttons, the layout of text, and background colours, among other elements.
3. Bootstrap:
    - Bootstrap, a popular front-end framework, is used to quickly implement responsive design, grids, and button styles. The framework ensures that the webpage is visually appealing and functions well on various screen sizes (desktops, tablets, smartphones).
4. JavaScript (JS)
    - Handling user input (reference string and frame count) and simulating Page replacement steps.
    - Dynamically generating HTML for the simulation steps and updating the result (hits, faults, and hit/fault rates).
    - JavaScript is used to implement the logic for simulating the FIFO, Optimal and LFU page replacement algorithms, dynamically updating the page with results, and managing user inputs.

Few snapshots of the project:



Welcome to Page Replacement Algorithms

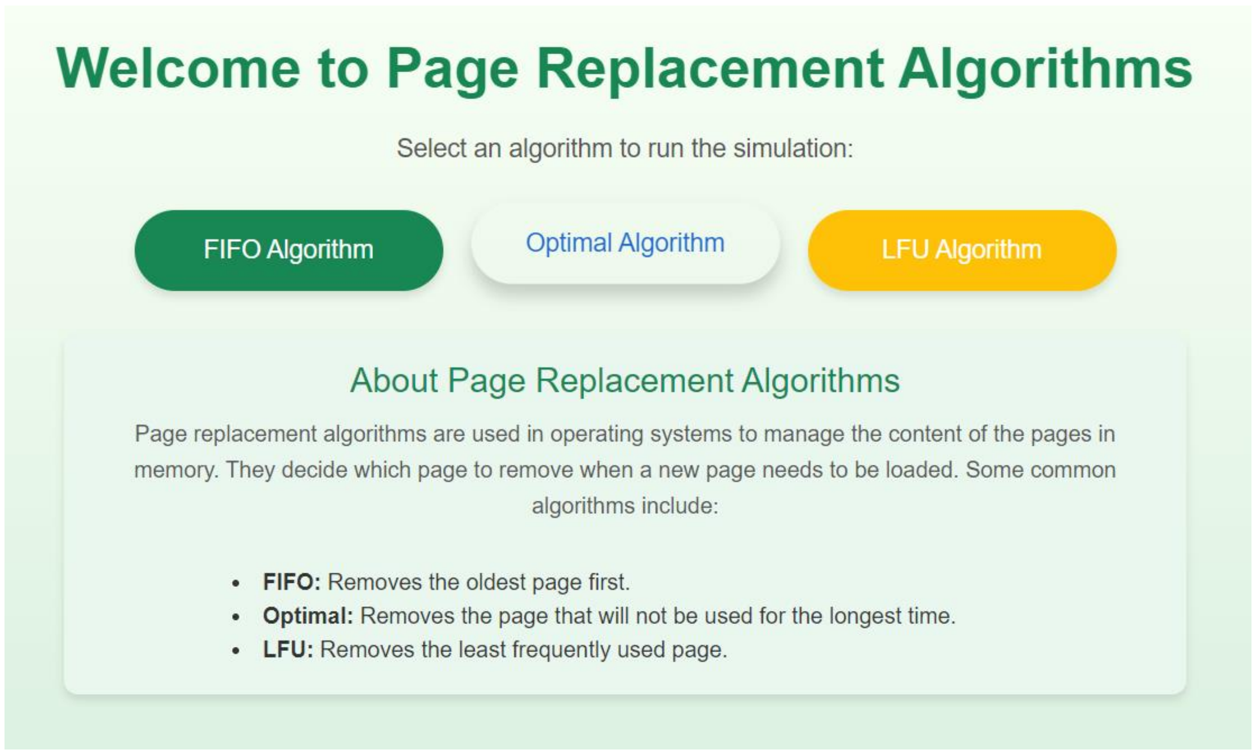
Select an algorithm to run the simulation:

About Page Replacement Algorithms

Page replacement algorithms are used in operating systems to manage the content of the pages in memory. They decide which page to remove when a new page needs to be loaded. Some common algorithms include:

- **FIFO:** Removes the oldest page first.
- **Optimal:** Removes the page that will not be used for the longest time.
- **LFU:** Removes the least frequently used page.

Fig 5.1a: Index page



Welcome to Page Replacement Algorithms

Select an algorithm to run the simulation:

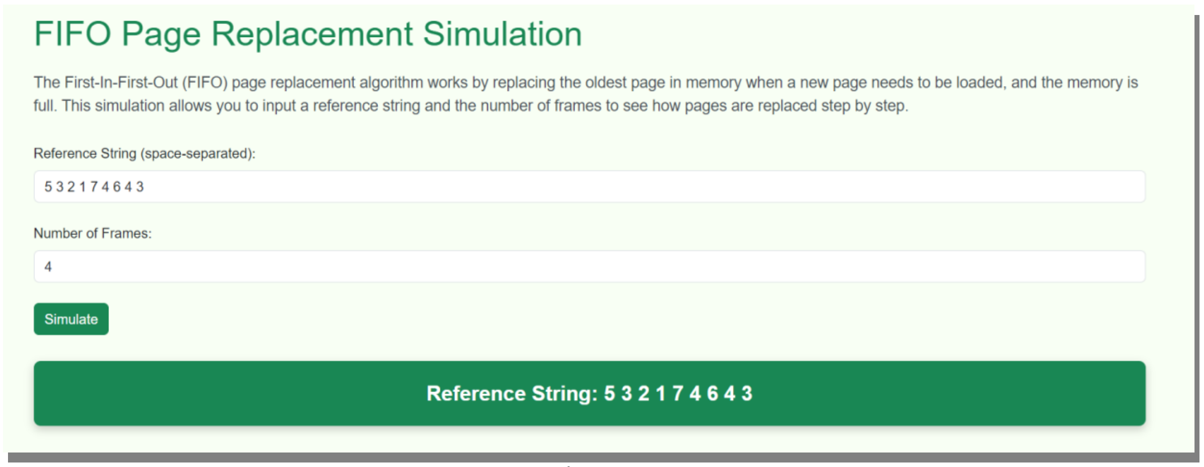
About Page Replacement Algorithms

Page replacement algorithms are used in operating systems to manage the content of the pages in memory. They decide which page to remove when a new page needs to be loaded. Some common algorithms include:

- **FIFO:** Removes the oldest page first.
- **Optimal:** Removes the page that will not be used for the longest time.
- **LFU:** Removes the least frequently used page.

Fig 5.1b: Hovering effect (on Optimal Algorithm)

1. FIFO Algorithm:



FIFO Page Replacement Simulation

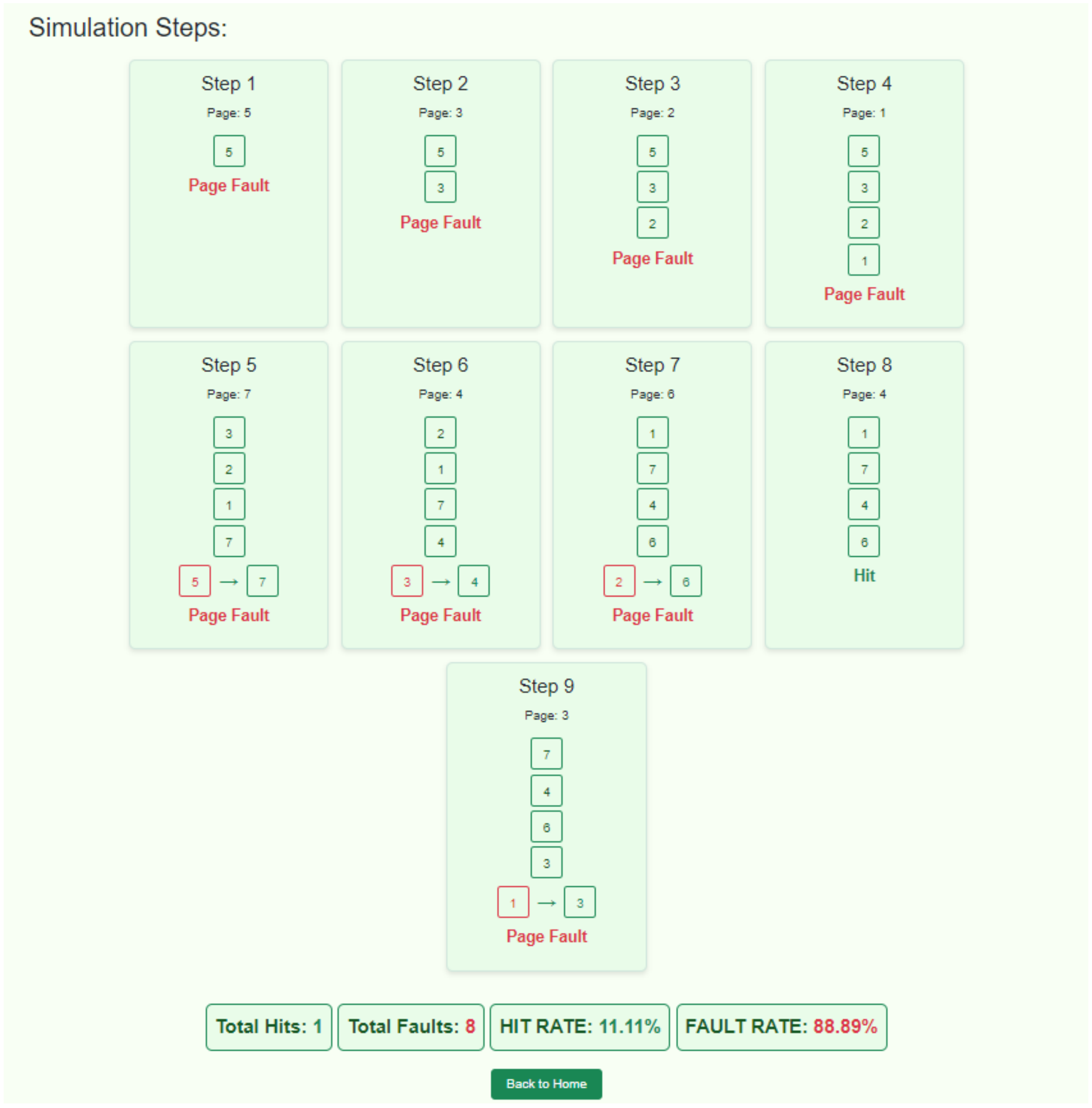
The First-In-First-Out (FIFO) page replacement algorithm works by replacing the oldest page in memory when a new page needs to be loaded, and the memory is full. This simulation allows you to input a reference string and the number of frames to see how pages are replaced step by step.

Reference String (space-separated): 5 3 2 1 7 4 6 4 3

Number of Frames: 4

Reference String: 5 3 2 1 7 4 6 4 3

Fig 5.2a



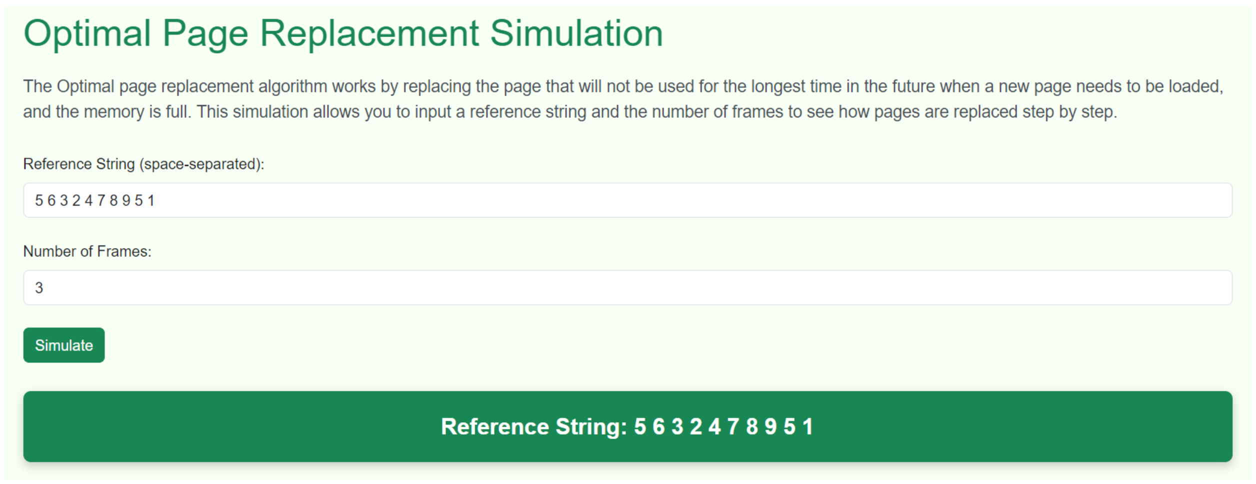
Simulation Steps:

Step	Page	Frames	Replacement	Result
Step 1	Page: 5	5		Page Fault
Step 2	Page: 3	5, 3		Page Fault
Step 3	Page: 2	5, 3, 2		Page Fault
Step 4	Page: 1	5, 3, 2, 1		Page Fault
Step 5	Page: 7	3, 2, 1, 7	5 → 7	Page Fault
Step 6	Page: 4	2, 1, 7, 4	3 → 4	Page Fault
Step 7	Page: 6	1, 7, 4, 6	2 → 6	Page Fault
Step 8	Page: 4	1, 7, 4, 6		Hit
Step 9	Page: 3	7, 4, 6, 3	1 → 3	Page Fault

Total Hits: 1 | Total Faults: 8 | HIT RATE: 11.11% | FAULT RATE: 88.89%

Fig 5.2b

2. Optimal Algorithm:



Optimal Page Replacement Simulation

The Optimal page replacement algorithm works by replacing the page that will not be used for the longest time in the future when a new page needs to be loaded, and the memory is full. This simulation allows you to input a reference string and the number of frames to see how pages are replaced step by step.

Reference String (space-separated):

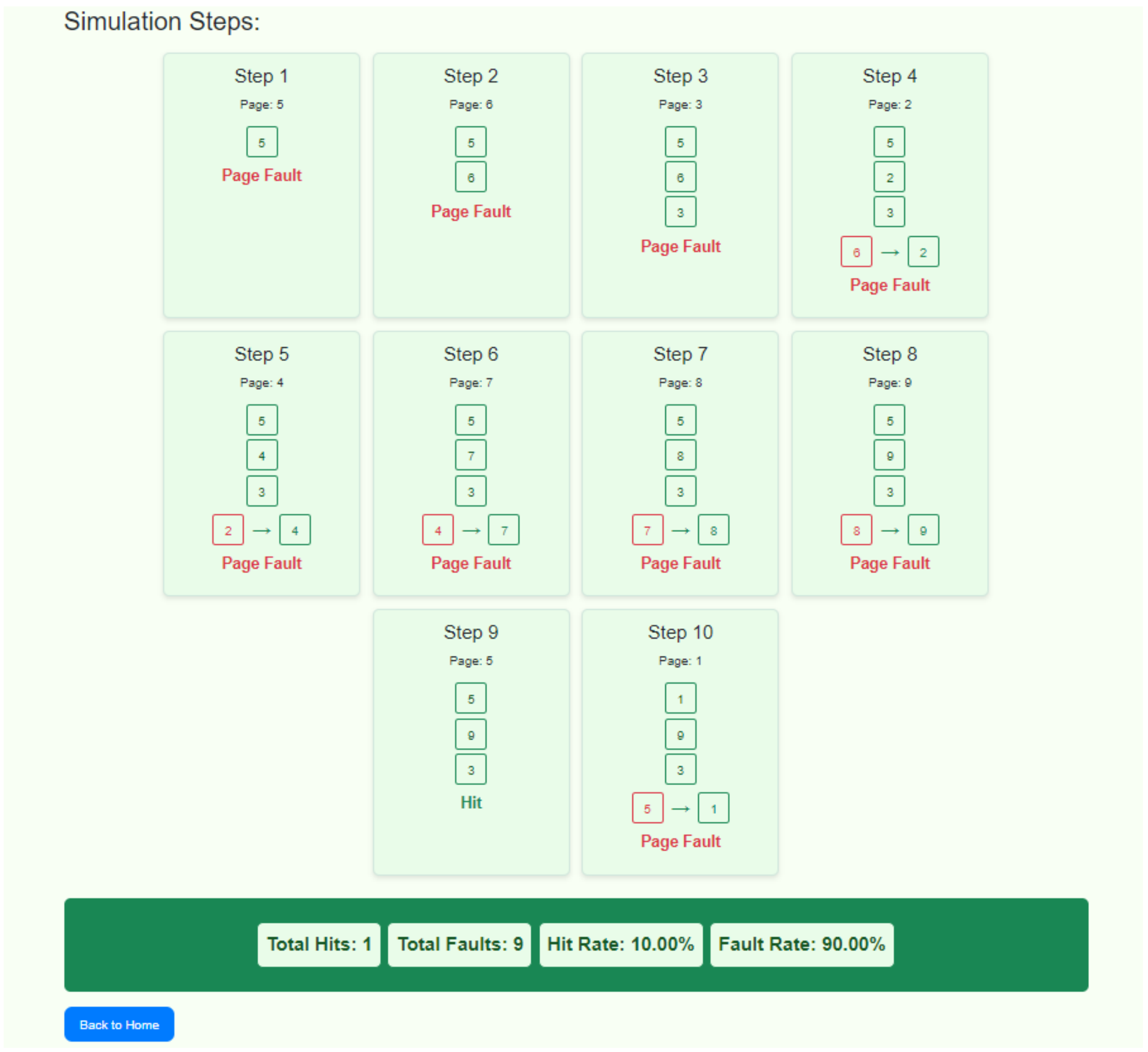
5 6 3 2 4 7 8 9 5 1

Number of Frames:

3

Reference String: 5 6 3 2 4 7 8 9 5 1

Fig 5.3a



Simulation Steps:

Step 1
Page: 5
5
Page Fault

Step 2
Page: 6
5
6
Page Fault

Step 3
Page: 3
5
6
3
Page Fault

Step 4
Page: 2
5
2
3
6 → 2
Page Fault

Step 5
Page: 4
5
4
3
2 → 4
Page Fault

Step 6
Page: 7
5
7
3
4 → 7
Page Fault

Step 7
Page: 8
5
8
3
7 → 8
Page Fault

Step 8
Page: 9
5
9
3
8 → 9
Page Fault

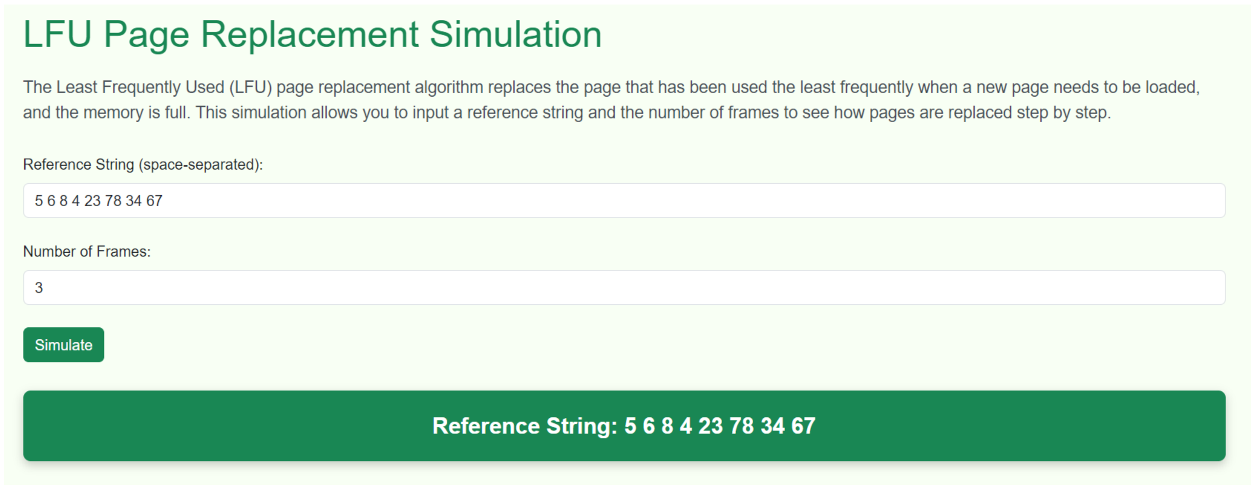
Step 9
Page: 5
5
9
3
Hit

Step 10
Page: 1
1
9
3
5 → 1
Page Fault

Total Hits: 1 Total Faults: 9 Hit Rate: 10.00% Fault Rate: 90.00%

Fig 5.3b

3. LFU Algorithm:



LFU Page Replacement Simulation

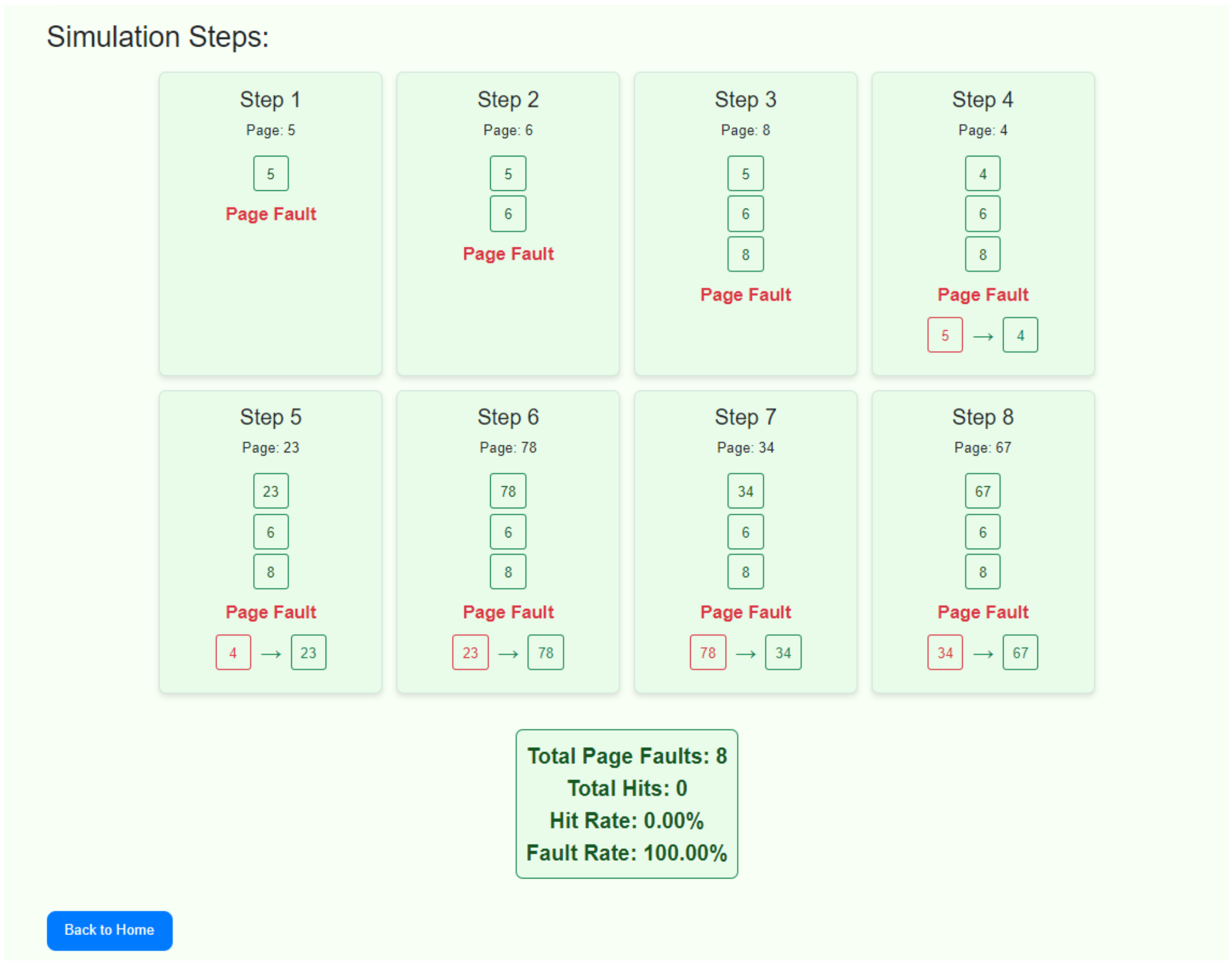
The Least Frequently Used (LFU) page replacement algorithm replaces the page that has been used the least frequently when a new page needs to be loaded, and the memory is full. This simulation allows you to input a reference string and the number of frames to see how pages are replaced step by step.

Reference String (space-separated): 5 6 8 4 23 78 34 67

Number of Frames: 3

Reference String: 5 6 8 4 23 78 34 67

Fig 5.4a



Simulation Steps:

```
Step 1  Page: 5   Frames: 5          Page Fault
Step 2  Page: 6   Frames: 5 6        Page Fault
Step 3  Page: 8   Frames: 5 6 8      Page Fault
Step 4  Page: 4   Frames: 4 6 8      Page Fault  5 → 4
Step 5  Page: 23  Frames: 23 6 8     Page Fault  4 → 23
Step 6  Page: 78  Frames: 78 6 8     Page Fault  23 → 78
Step 7  Page: 34  Frames: 34 6 8     Page Fault  78 → 34
Step 8  Page: 67  Frames: 67 6 8     Page Fault  34 → 67
```

Total Page Faults: 8
Total Hits: 0
Hit Rate: 0.00%
Fault Rate: 100.00%

Fig 5.4b

Conclusion

In this project, we explored and simulated various page replacement algorithms used in operating systems to efficiently manage memory. Through the implementation and analysis of First-In-First-Out (FIFO), Least Recently Used (LRU), and Optimal Page Replacement algorithms, we were able to gain insights into their functionality, strengths, and weaknesses in handling page faults and optimizing memory usage.

The simulation allowed us to observe how each algorithm responds to different memory access patterns, calculating the hit rate and fault rate to measure the efficiency of each approach. By comparing the number of page faults and hits across various scenarios, we identified how different algorithms perform in terms of both theoretical efficiency (Optimal) and practical applicability (FIFO and LRU).

Key takeaways from the project include:

- FIFO is simple but may not always provide optimal performance, as it tends to suffer from Belady's anomaly, where increasing memory frames can sometimes increase the number of page faults.
- LRU, though more efficient in many real-world scenarios, requires more complex tracking of page usage, but it significantly reduces page faults compared to FIFO.
- The Optimal algorithm, while theoretically the best, is not feasible for real-world systems due to its requirement for future knowledge of page references.

The project also highlighted the importance of page replacement strategies in managing virtual memory, and the impact of these strategies on system performance, particularly in preventing thrashing and optimizing memory usage.

Ultimately, the interactive simulation provided an intuitive and visual way to understand these complex operating system concepts, making it an effective tool for learning and experimenting with memory management strategies. This project not only deepened our understanding of memory management but also underscored the need for efficient page replacement algorithms in ensuring that systems run optimally, even with limited memory resources.