LOVELY PROFESSIONAL UNIVERSITY

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

COURSE: CSE 316
OPERATING SYSTEM

SUBMITTED TO: **Mr. Kundan**

TOPIC

"Efficient Page Replacement Algorithm Simulator"

A.	Full name	**Priyansi Sahu**
	Reg No.	**12417106**
	Roll No	**48**
B.	Full name	**Mandeep Kumar Roshan**
	Reg No.	**12417125**
	Roll No	**49**
C.	Full name	**Shivam Sinha**
	Reg No.	**12418397**
	Roll No	**50**

Table of Contents

1. Project Overview

Introduction

In Operating Systems, memory management is one of the most important concepts because it decides how efficiently a computer runs multiple programs. When the number of physical frames is limited and new pages need to be loaded, the operating system must choose which existing page to remove. This decision is made using page replacement algorithms.

This project focuses on building a simple and interactive simulator that demonstrates how three popular page replacement algorithms—FIFO, LRU, and Optimal—work in real situations. The simulator takes input from the user and shows step-by-step how the frames change, when page faults occur, and how many hits and misses each algorithm produces. A graphical interface built using Tkinter makes the simulation easy to understand. Overall, this project provides a practical way to learn and visualize memory management instead of only reading theory.

Objective of the Project

1. To simulate FIFO, LRU, and Optimal page replacement algorithms
2. To allow users to enter frames and reference string as input.
3. To display step-by-step frame updates for each algorithm.
4. To compare the performance of different algorithms using page faults, hits, and hit ratio.
5. To create a simple and user-friendly GUI for visualization.
6. To help students clearly understand how different algorithms behave for the same input.

Scope of the Project:

The scope of this project includes implementing three page replacement algorithms and visualizing their behavior using a graphical interface. The simulator works for any valid reference string and frame count entered by the user. It focuses on educational understanding rather than system-level performance. The project is suitable for OS lab assignments, demonstrations, and learning memory management concepts.

Future enhancements like adding more algorithms, charts, exporting results, or creating a web-based version are possible, but the current scope is limited to algorithm simulation, performance comparison, and GUI-based representation.

2. Module-Wise Breakdown

The **Efficient Page Replacement Algorithm Simulator** is organized into several key modules, each responsible for handling a specific functionality within the system. Below is a breakdown of the major modules:

---

1. Graphical User Interface (GUI) Module

Purpose:

- Provides an easy-to-use interface for entering the number of frames and the reference string.
- Displays simulation results for FIFO, LRU, and Optimal algorithms in a readable format.
- Offers a clean visualization of step-by-step page loading and replacement.

Features:

- Simple and user-friendly input fields for frames and reference string.
- A "Run Simulation" button that triggers all three algorithms.
- scrollable text output area that clearly presents:
    - Page being accessed
    - Current frame contents
    - Hit or Miss for each step
- Automatically highlights total page faults, hit ratio, and the best-performing algorithm.
- Error pop-ups for invalid inputs using message boxes.

---

2. Algorithm Processing & Simulation Module

Purpose:

- Executes the FIFO, LRU, and Optimal algorithms step-by-step.
- Tracks frame states, page faults, and hit/miss data throughout the simulation.
- Ensures correct page replacement logic according to each algorithm's rules.

Features:

- Processes the reference string sequentially and maintains internal data structures.
- Records every state change such as:
    - Which page enters the frame
    - Which page gets replaced
    - When a hit or miss occurs
- Calculates:

    - Total page faults
    - Total hits
    - Hit ratio
- Uses algorithm-specific logic:
    - FIFO uses queue-based ordering
    - LRU tracks recent usage
    - Optimal predicts future references
- Returns detailed structured results back to the GUI for display.

---

3. Result Visualization & Performance Comparison Module

Purpose:

- Converts raw simulation outputs into clear, readable results for the user.
- Helps compare all three algorithms based on performance metrics.
- Enhances understanding by summarizing the outcome.

Features:

- Automatically identifies and displays the **best algorithm** (lowest page faults).
- Ensures that students can easily compare FIFO, LRU, and Optimal under the same input.
- Shows total metrics at the end of each simulation:
    - Total page faults
    - Total hits
    - Hit ratio
- Provides clean formatting of steps for each algorithm:
    - Step number
    - Current page
    - Frame status
    - Hit/Miss tag

---

4. Future Extension Module

Purpose:

- Represents the additional upgrades that can be added in future versions of the simulator.
- Encourages further development beyond the basic implementation.

Features:

- Graphical charts (bar graphs, line graphs) comparing page faults visually.
- Add advanced algorithms like LFU, MFU, or Second Chance.
- Include an option to export simulation results as PDF or text file.
- Implement a web-based version using Flask or Streamlit for online use.
- Add support for automatic test-case generation and batch simulations.

3. Functionalities

The **Efficient Page Replacement Algorithm Simulator** is designed to help users test, observe, and compare the behavior of different page replacement algorithms in a simple and interactive way. Below are the key functionalities of the simulator:

1. Page Replacement Simulation

- Executes FIFO, LRU, and Optimal page replacement algorithms.
- Processes the reference string step-by-step and updates frame contents.
- Identifies hits and misses for each page access.
- Calculates total page faults, total hits, and hit ratio for each algorithm.

2. Frame Visualization Module

- Displays the current state of frames after each page request.
- Shows which page is inserted or replaced.
- Indicates Hit or Miss for every step.
- Presents frame transitions in an easy-to-read format.

3. User Input and Validation

- Accepts number of frames and the reference string from the user.
- Ensures the reference string contains only integers.
- Shows error messages for invalid or empty inputs.
- Prevents incorrect simulation runs by validating inputs before processing.

4. Performance Comparison

- Automatically identifies the best-performing algorithm.
- Compares FIFO, LRU, and Optimal based on page faults and hit ratio
- Helps users easily understand how algorithm efficiency varies with input.

5. Graphical User Interface (GUI)

- Provides a simple and intuitive interface using Tkinter.
- Includes clean input fields and a Run Simulation button.
- Contains a scrollable output area for long results.
- Allows users to run the simulator without command-line knowledge.

6. Error Handling and User Feedback

- Shows alerts for non-integer input, empty fields, or invalid frame count.
- Prevents crashes by handling incorrect inputs safely.
- Ensures smooth and user-friendly simulation

4. Technology Used

The **Efficient Page Replacement Algorithm Simulator** is developed using a combination of programming languages, libraries, and tools that support algorithm execution, visualization, and user interaction. The technologies used in the project are listed below:

1. Programming Languages

- **Python** – Used for implementing the page replacement algorithms (FIFO, LRU, Optimal), handling input processing, performing step-by-step simulation, and generating the final output. Python was chosen because of its readability and suitability for educational simulations.

2. Libraries and Tools

- **Tkinter** – Used for building the Graphical User Interface (GUI). It allows users to enter frames and reference strings easily and displays simulation results in a clear, scrollable window.
- **Python Standard Libraries** – Lists, dictionaries, and control structures were used to implement page replacement logic without external dependencies.
- **Messagebox (Tkinter)** - Used to show error messages for invalid input, improving user experience.
- **ScrolledText (Tkinter)** - Displays long simulation results in a scrollable text area.

3. Development & Version Control Tools

- **Git & GitHub** – Used for maintaining project revisions, creating multiple commits, managing branches, and storing the complete project repository.
- **VS Code** – Used as the main IDE for writing, debugging, and running Python code.
- **Python Interpreter** – Required to execute both the simulation logic and the GUI application.

5. Flow Diagram

Flowchart Representation

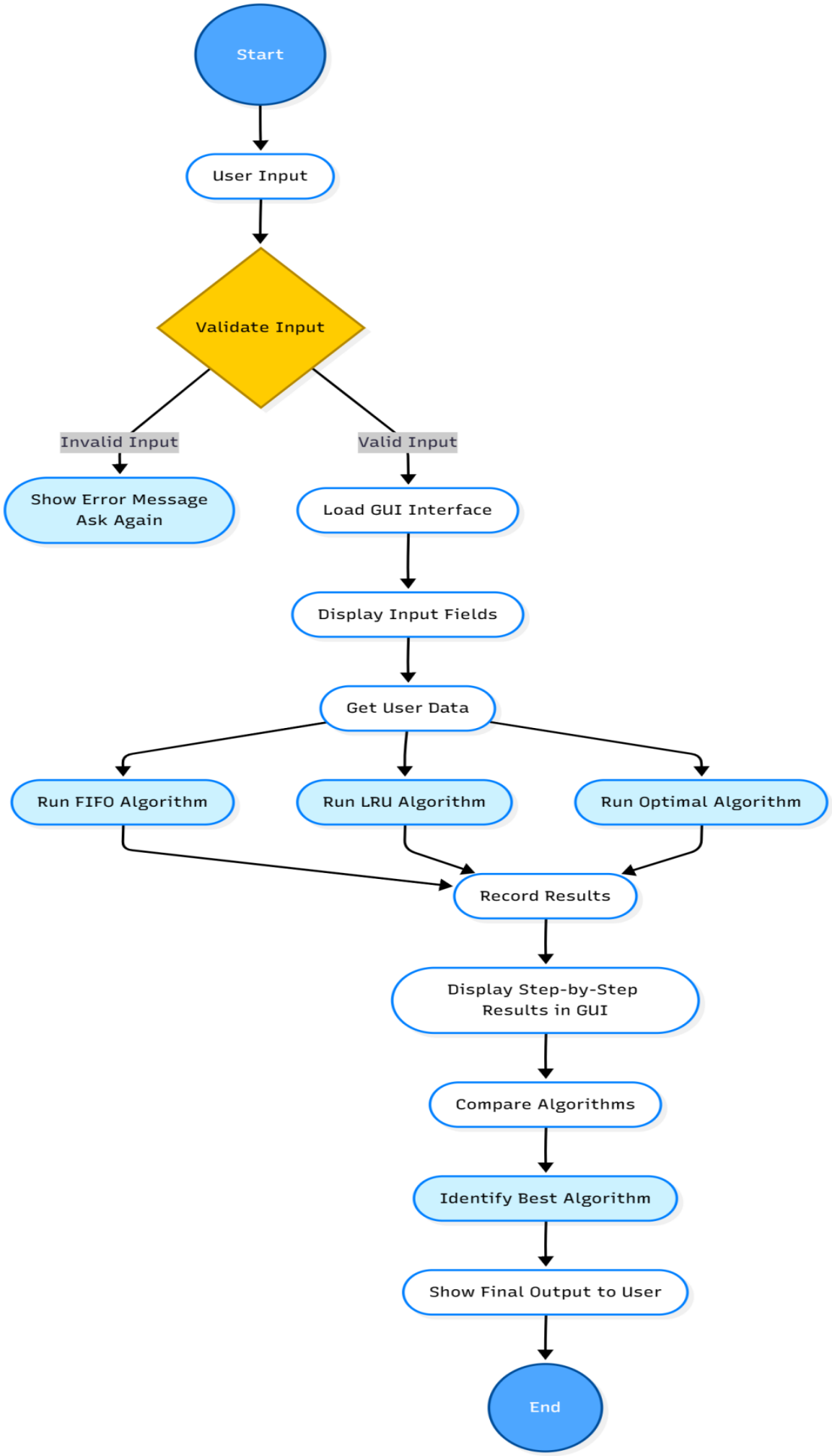

Figure 1: Flow Diagram of Page Replacement Simulator

6. GUI Screenshots

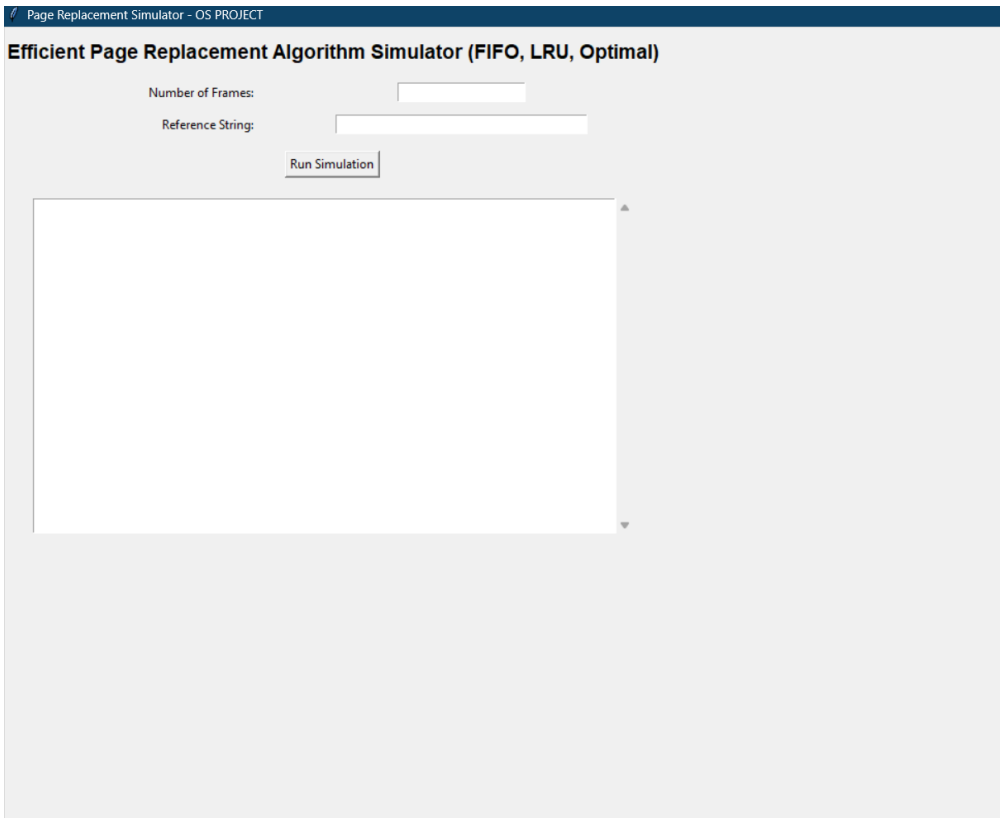

Figure 2: GUI Interface for Entering Frames and Reference String

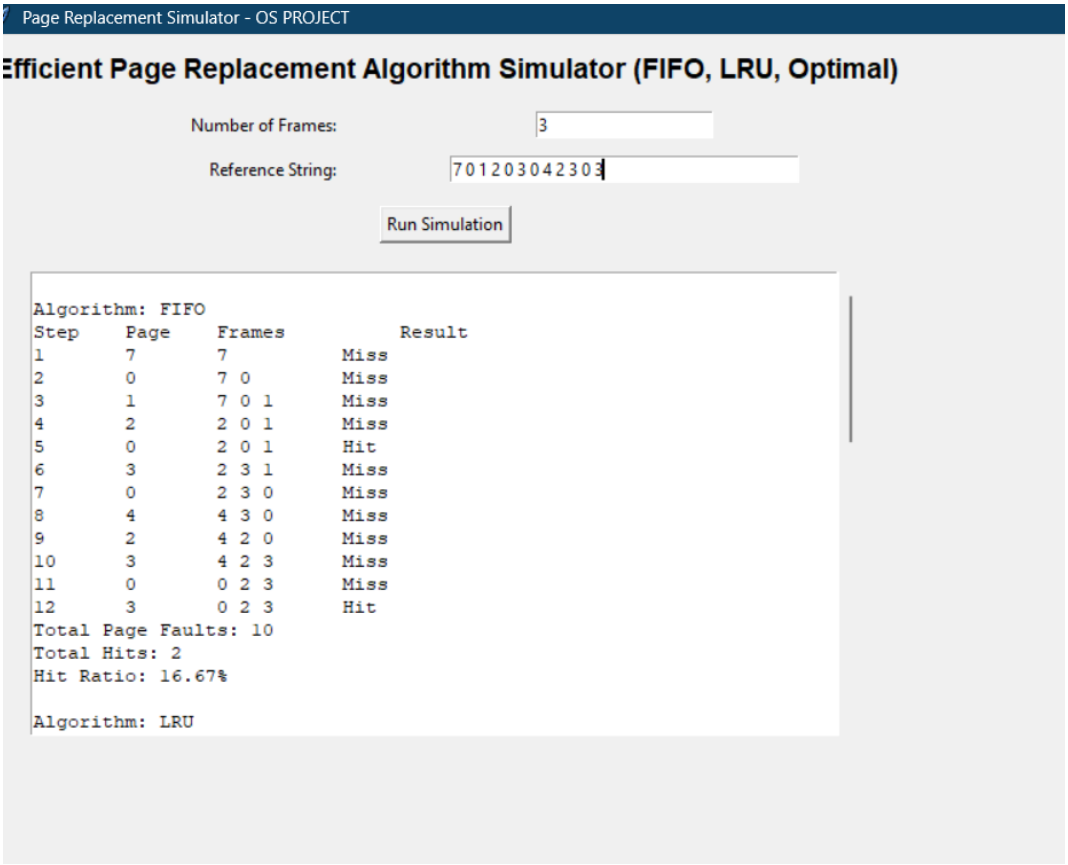

Figure 3: GUI Interface and FIFO Simulation Output for Test Case 1

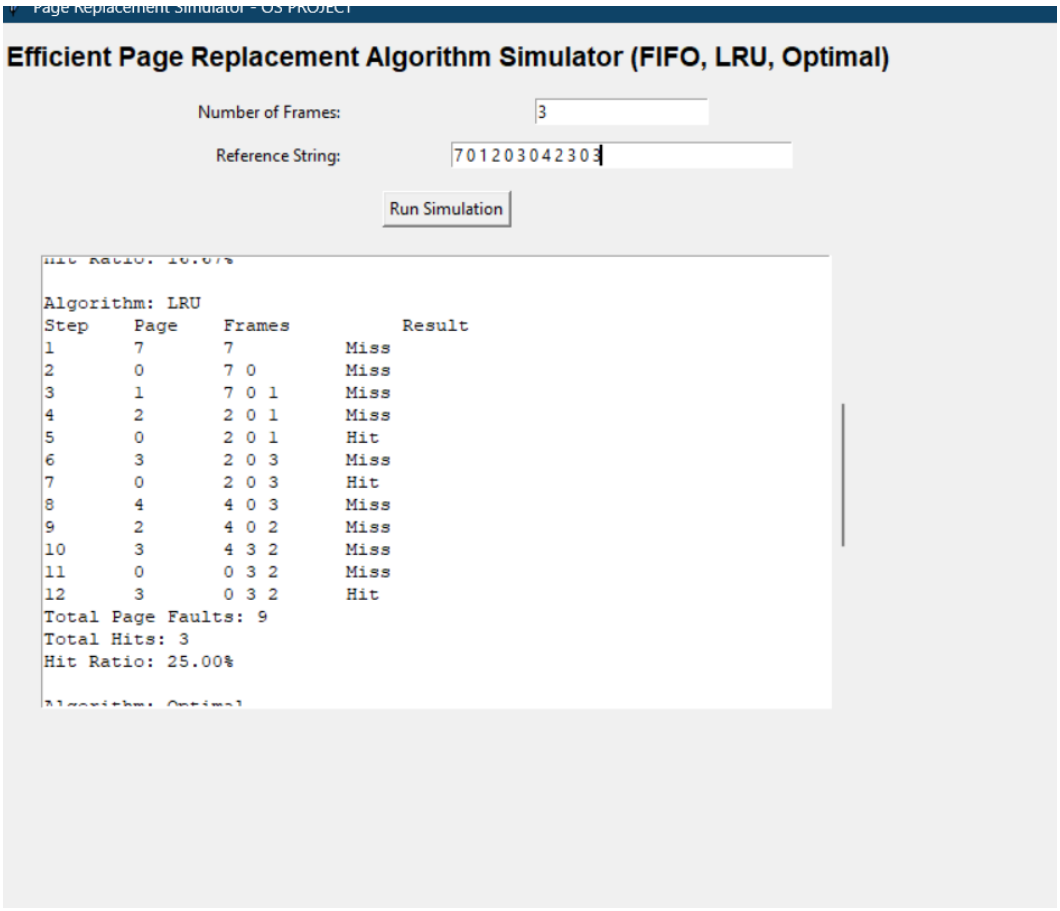

Figure 4: LRU Algorithm Output for Test Case 1

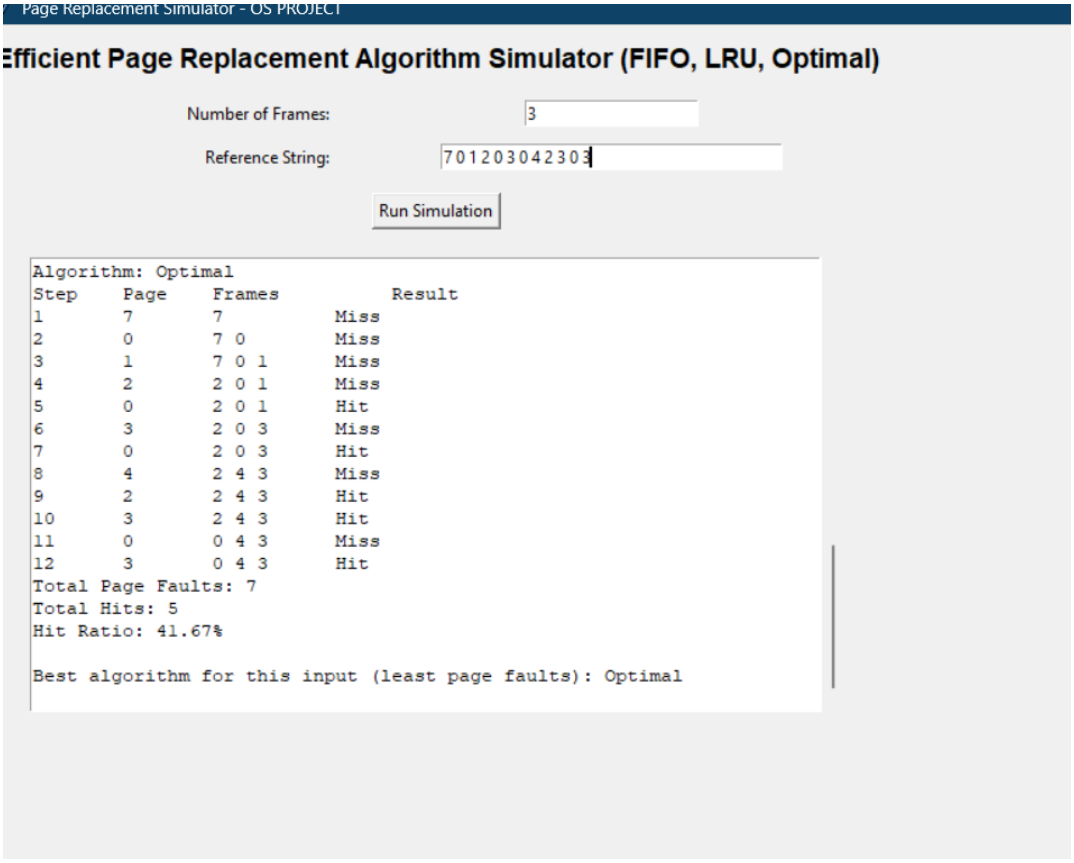

Figure 5: Optimal Algorithm Output Showing Best Algorithm Selection for Test Case 1

7. Revision Tracking on GitHub

Repository Name:

GitHub Link:

8. Conclusion and Future Scope

Conclusion:

The **Efficient Page Replacement Algorithm Simulator** successfully demonstrates the working of FIFO, LRU, and Optimal page replacement algorithms in an interactive and easy-to-understand manner. The simulator helps users visualize how pages are loaded into frames, how page faults occur, and how different algorithms behave under the same conditions. By integrating Python and Tkinter, the tool offers a simple and user-friendly interface that allows users to experiment with various inputs without technical complexity.
Overall, the project strengthens conceptual understanding of memory management and provides a clear comparison of algorithm performance, making it highly effective for academic learning and operating systems demonstrations.

Future Scope:

1. Addition of More Algorithms

- Extend the simulator by adding LFU, MFU, Second-Chance, and Clock algorithms to support wider comparison.

2. Graphical Performance Visualization

- Integrate bar graphs or line charts to visually compare page faults and hit ratios across algorithms.

3. Web-Based or Mobile Version

- Create an online or mobile-friendly version using frameworks like Flask or Streamlit so users can access it without installing Python.

4. Export and Reporting Features

- Add the ability to export simulation results as PDF, text, or image files for documentation and academic submissions.

5. Batch Simulation Mode

- Implement a feature where multiple reference strings can be tested automatically to evaluate performance across datasets.

6. Intelligent Input Suggestions

- Use basic analysis or heuristics to suggest challenging reference strings for better testing of algorithm behavior.

9. References

Online Resources

- **Python Official Documentation** – Used for understanding Python syntax, standard libraries, and functions used in the project. https://docs.python.org/3/
- **Tkinter Official Documentation** –Referred for building the GUI components such as buttons, labels, message boxes, and scrollable text fields. https://docs.python.org/3/library/tkinter.html
- **GeeksforGeeks** – Used to study the theoretical concepts of FIFO, LRU, and Optimal page replacement algorithms. https://www.geeksforgeeks.org/page-replacement-algorithms-in-operating-systems/
- **Wikipedia** – Used for general reference on page replacement strategies and OS memory management fundamentals. https://en.wikipedia.org/wiki/Page_replacement_algorithm
- **Stack Overflow** – Checked for minor troubleshooting related to Python GUI behavior and input validation techniques used in Tkinter. https://stackoverflow.com/

Appendix

A. Appendix A – AI-Generated Project Elaboration / Breakdown Report

This appendix contains the AI-generated breakdown of the project, which was used to plan the structure, workflow, and implementation of the *Efficient Page Replacement Algorithm Simulator*. The AI breakdown helped in understanding the overall requirements and guided the development of the GUI, algorithm logic, and visualization modules.

Project Overview

The Efficient Page Replacement Algorithm Simulator is designed to demonstrate how different page replacement algorithms behave under varying memory constraints. The project focuses on three standard algorithms used in Operating Systems: FIFO (First-In-First-Out), LRU (Least Recently Used), and Optimal Page Replacement.

The simulator accepts input from the user such as the number of frames and a page reference string. It then processes this input through each algorithm and generates step-by-step output showing the state of frames, hits and misses, and total page faults. The implementation uses Python and Tkinter to build a user-friendly GUI that makes the simulation interactive and easy to follow. The goal is to strengthen conceptual understanding of memory management by visualizing how algorithms work internally.

Problem Understanding and Motivation

Page replacement is a critical part of memory management. When the physical memory (frames) is full and a new page is needed, the operating system must decide which existing page to replace. Different strategies produce different performance results. Students often struggle to understand these algorithms from theory alone.

This project aims to solve this challenge by presenting the algorithms visually. Instead of reading abstract definitions, learners can observe how pages move in and out of frames, how page faults occur, and why one algorithm performs better than another in specific scenarios. The simulator enhances conceptual clarity through practice-oriented visualization.

System Requirements and Design Considerations

Before starting the implementation, several requirements and constraints were analyzed:

Functional Requirements

- The system must allow the user to enter the number of frames.
- The system must accept a reference string consisting of integers.
- The simulator should run FIFO, LRU, and Optimal algorithms on the same input.
- The GUI must display step-by-step execution of all algorithms.
- The final output must include total page faults, hits, and hit ratio.

Non-Functional Requirements

- The interface should be simple and intuitive.
- Execution should be fast and responsive for most input sizes.
- The code should be modular and maintainable.

Design Constraints

- Only Python and Tkinter should be used (as per project scope).
- The application should be light, without heavy external libraries.
- The system should run on any machine with Python installed.

Module-Wise Breakdown

The complete project is divided into the following interconnected modules:

A. GUI Module

This module handles all user interactions. It provides input boxes for frame count and reference string, buttons to start simulation, and a scrollable display area for the results. The GUI also shows error messages in case of invalid inputs.

B. Algorithm Simulation Module

This is the core logic component. It contains the implementation of FIFO, LRU, and Optimal algorithms. It processes the input page sequence, updates frame contents at every step, detects hits/misses, and calculates total page faults.

C. Visualization and Output Module

This module formats and presents the results generated by the algorithm module. It shows each step in a readable format and highlights the best-performing algorithm based on page faults.

Algorithm Design Explanation

FIFO (First-In-First-Out)

- The oldest loaded page is removed when replacement is needed.
- Simple to implement but not always optimal.
- Works well for sequential data patterns.

LRU (Least Recently Used)

- Removes the page that has not been used for the longest time.
- Requires tracking recent usage of pages.
- More accurate than FIFO in practical scenarios.

Optimal Algorithm

- Replaces the page that will not be used for the longest period in the future.
- Produces the least number of page faults.
- Not practical for real-time systems but valuable for comparison.

Each algorithm generates a step-by-step representation showing which page is being inserted, which page is removed (if any), and whether there is a hit or miss.

Data Flow Explanation

1. User inputs the number of frames and reference string.
2. The system validates the input.
3. Each algorithm processes the reference string independently.
4. Frame contents are updated at every page access.
5. The simulator records hits, misses, and page faults.
6. Results are visualized through the GUI.
7. A performance comparison is shown at the end.

This structured workflow ensures clarity in understanding each algorithm's internal functioning.

Technology Breakdown

Python

Used for implementing the logic of the algorithms and for handling all operations.

Tkinter

Used to build the GUI interface, manage user inputs, and display results.

VS Code

Used for writing, testing, and debugging the code.

GitHub

Maintains revision history, branches, and backups of the project.

Expected Outcomes and Learning Goals

By completing this project, the student gains:

- A clear understanding of FIFO, LRU, and Optimal algorithms.
- Ability to visualize memory management techniques.
- Experience in GUI development using Tkinter.
- Hands-on understanding of page faults, hits, and frame replacement.
- Knowledge of how to compare algorithm performance.
- Skills in writing modular, documented, and maintainable code.

Conclusion of Breakdown

This AI-generated breakdown serves as a structured guide for understanding the full design, development, and purpose of the Efficient Page Replacement Algorithm Simulator. It provides a comprehensive explanation of every major part of the project, from problem analysis to implementation flow.
The breakdown ensures that each component is logically connected and well-understood before writing code or documenting final outcomes.

B. Appendix B – Problem Statement

Problem Title: "*Efficient Page Replacement Algorithm Simulator"*

Problem Statement:
The objective of this project is to design and implement a simulator that compares different page replacement algorithms used in memory management. The simulator must accept user inputs such as the number of frames and a page reference string.
It should then run FIFO, LRU, and Optimal algorithms on the given input and display step-wise updates of frame contents, hits, misses, and page faults through a graphical user interface. Additionally, the simulator should compute performance metrics such as hit ratio and determine which algorithm performs best for the given input.
The purpose of the project is to help students understand page replacement behavior in operating systems through practical visualization rather than only theoretical study.

C. Appendix C – Code

 # page_replacement.py

```
def simulate_fifo(pages, frame_count):
    frames = []
    page_faults = 0
    hits = 0
    steps = []
    queue_index = 0  # to track which frame to replace

    for page in pages:
        step_info = {}
        if page in frames:
            hits += 1
            hit_or_miss = "Hit"
```

```
        else:
            page_faults += 1
            hit_or_miss = "Miss"
            if len(frames) < frame_count:
                frames.append(page)
            else:
                # replace in FIFO manner
                frames[queue_index] = page
                queue_index = (queue_index + 1) % frame_count

        step_info["page"] = page
        step_info["frames"] = frames.copy()
        step_info["result"] = hit_or_miss
        steps.append(step_info)

    total_access = len(pages)
    hit_ratio = hits / total_access if total_access > 0 else 0

    return {
        "name": "FIFO",
        "steps": steps,
        "page_faults": page_faults,
        "hits": hits,
        "hit_ratio": hit_ratio
    }


def simulate_lru(pages, frame_count):
    frames = []
```

```
page_faults = 0
hits = 0
steps = []

# to track last used index of each page in frames
last_used = {}

for current_index, page in enumerate(pages):
    step_info = {}

    if page in frames:
        hits += 1
        hit_or_miss = "Hit"
    else:
        page_faults += 1
        hit_or_miss = "Miss"

        if len(frames) < frame_count:
            frames.append(page)
        else:
            # find least recently used page
            # check last_used index for each page in frames
            lru_page = None
            lru_index = float("inf")

            for p in frames:
                # if never used before, treat as very old
                idx = last_used.get(p, -1)
                if idx < lru_index:
```

```
                lru_index = idx
                lru_page = p

            # replace lru_page with current page
            replace_index = frames.index(lru_page)
            frames[replace_index] = page

        # update last used index of current page
        last_used[page] = current_index

        step_info["page"] = page
        step_info["frames"] = frames.copy()
        step_info["result"] = hit_or_miss
        steps.append(step_info)

    total_access = len(pages)
    hit_ratio = hits / total_access if total_access > 0 else 0

    return {
        "name": "LRU",
        "steps": steps,
        "page_faults": page_faults,
        "hits": hits,
        "hit_ratio": hit_ratio
    }


def simulate_optimal(pages, frame_count):
    frames = []
```

```
page_faults = 0
hits = 0
steps = []

for i, page in enumerate(pages):
    step_info = {}

    if page in frames:
        hits += 1
        hit_or_miss = "Hit"
    else:
        page_faults += 1
        hit_or_miss = "Miss"

        if len(frames) < frame_count:
            frames.append(page)
        else:
            # find page to replace using optimal strategy
            farthest_index = -1
            page_to_replace = None

            for p in frames:
                # check when this page will appear next
                try:
                    next_use = pages.index(p, i + 1)
                except ValueError:
                    # page not used again, best to replace this
                    next_use = float("inf")
```

```
                if next_use > farthest_index:
                    farthest_index = next_use
                    page_to_replace = p

            replace_index = frames.index(page_to_replace)
            frames[replace_index] = page

        step_info["page"] = page
        step_info["frames"] = frames.copy()
        step_info["result"] = hit_or_miss
        steps.append(step_info)

    total_access = len(pages)
    hit_ratio = hits / total_access if total_access > 0 else 0

    return {
        "name": "Optimal",
        "steps": steps,
        "page_faults": page_faults,
        "hits": hits,
        "hit_ratio": hit_ratio
    }


def print_result(result):
    print(f"\nAlgorithm: {result['name']}")
    print("Step\tPage\tFrames\t\tResult")
    for idx, step in enumerate(result["steps"], start=1):
        frames_str = " ".join(str(x) for x in step["frames"])
```

```
        print(f"{idx}\t{step['page']}\t{frames_str:<10}\t{step['result']}")
    print(f"Total Page Faults: {result['page_faults']}")
    print(f"Total Hits: {result['hits']}")
    print(f"Hit Ratio: {result['hit_ratio']*100:.2f}%")


def main():
    print("Efficient Page Replacement Algorithm Simulator")
    frame_count = int(input("Enter number of frames: "))
    ref_string = input("Enter reference string (space-separated pages): ")
    pages = list(map(int, ref_string.split()))

    fifo_result = simulate_fifo(pages, frame_count)
    lru_result = simulate_lru(pages, frame_count)
    optimal_result = simulate_optimal(pages, frame_count)

    print_result(fifo_result)
    print_result(lru_result)
    print_result(optimal_result)

    # simple comparison
    all_results = [fifo_result, lru_result, optimal_result]
    best = min(all_results, key=lambda r: r["page_faults"])
    print(f"\nBest algorithm for this input (least page faults): {best['name']}")


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

 # gui.py

```
import tkinter as tk
from tkinter import messagebox, scrolledtext

from page_replacement import simulate_fifo, simulate_lru, simulate_optimal


def run_simulation():
    try:
        frame_count = int(entry_frames.get())
    except ValueError:
        messagebox.showerror("Error", "Please enter a valid integer for frames.")
        return

    ref_string = entry_ref.get().strip()
    if not ref_string:
        messagebox.showerror("Error", "Please enter a reference string.")
        return

    try:
        pages = list(map(int, ref_string.split()))
    except ValueError:
        messagebox.showerror("Error", "Reference string must contain only
integers separated by spaces.")
        return

    fifo_result = simulate_fifo(pages, frame_count)
    lru_result = simulate_lru(pages, frame_count)
    optimal_result = simulate_optimal(pages, frame_count)

    # clear previous output
    text_output.delete(1.0, tk.END)

    for result in [fifo_result, lru_result, optimal_result]:
```

```
        text_output.insert(tk.END, f"\nAlgorithm: {result['name']}\n")
        text_output.insert(tk.END, "Step\tPage\tFrames\t\tResult\n")
        for idx, step in enumerate(result["steps"], start=1):
            frames_str = " ".join(str(x) for x in step["frames"])
            text_output.insert(
                tk.END,
                f"{idx}\t{step['page']}\t{frames_str:<10}\t{step['result']}\n"
            )
        text_output.insert(tk.END, f"Total Page Faults: {result['page_faults']}\n")
        text_output.insert(tk.END, f"Total Hits: {result['hits']}\n")
        text_output.insert(tk.END, f"Hit Ratio: {result['hit_ratio']*100:.2f}%\n")

    all_results = [fifo_result, lru_result, optimal_result]
    best = min(all_results, key=lambda r: r["page_faults"])
    text_output.insert(tk.END, f"\nBest algorithm for this input (least page
faults): {best['name']}\n")



# GUI setup
root = tk.Tk()
root.title("Page Replacement Simulator - OS PROJECT")

label_title = tk.Label(root, text="Efficient Page Replacement Algorithm
Simulator (FIFO, LRU, Optimal)", font=("Arial", 14, "bold"))
label_title.grid(row=0, column=0, columnspan=2, pady=10)

label_frames = tk.Label(root, text="Number of Frames:")
label_frames.grid(row=1, column=0, sticky="e", padx=5, pady=5)

entry_frames = tk.Entry(root)
entry_frames.grid(row=1, column=1, padx=5, pady=5)

label_ref = tk.Label(root, text="Reference String:")
label_ref.grid(row=2, column=0, sticky="e", padx=5, pady=5)

entry_ref = tk.Entry(root, width=40)
entry_ref.grid(row=2, column=1, padx=5, pady=5)
```

```
button_run = tk.Button(root, text="Run Simulation", command=run_simulation)
button_run.grid(row=3, column=0, columnspan=2, pady=10)

text_output = scrolledtext.ScrolledText(root, width=70, height=20)
text_output.grid(row=4, column=0, columnspan=2, padx=10, pady=10)

root.mainloop()
```