

Efficient Page Replacement Algorithm Simulator

**Submitted to: -**  **Dr. Parvinder Singh**

**STUDENT(S):**

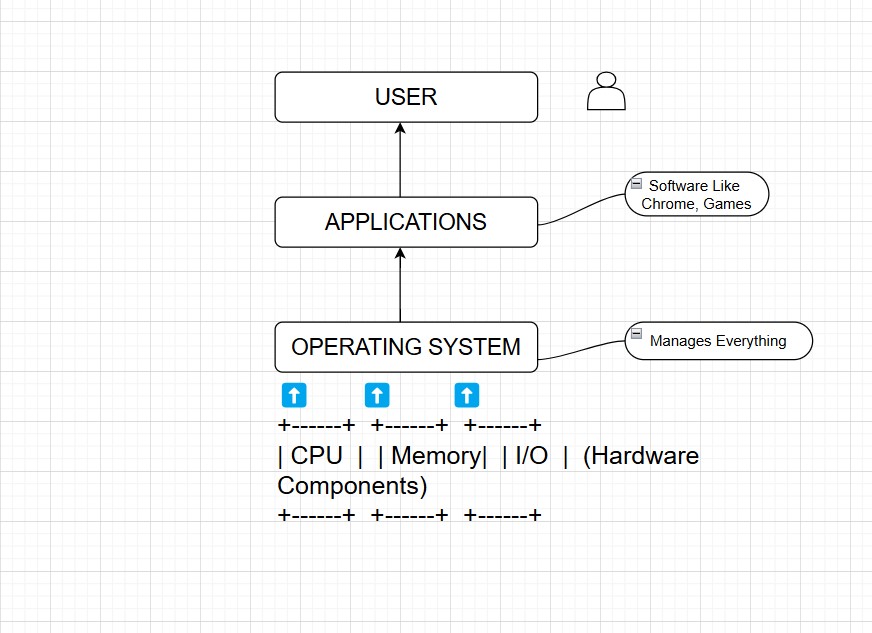
|  |  |  |
| --- | --- | --- |
| A. | Full name | Ushman Khan |
|  | Roll No. | 20 |
|  | Section | K23DF |
|  | UID/Registration number | 12303005 |

Introduction to OS.

An Operating System (OS) is the software that manages all hardware and software on a computer. It acts as a bridge between the user and the computer. Without an OS, a computer is just a box of hardware with no instructions on how to function. What does an OS do?

1. Manages Hardware – Controls CPU, memory, storage, and input/output devices.
2. Runs Applications – Allows software like browsers, games, and media players to function.
3. File Management – Helps store, organize, and retrieve data.
4. Security & User Control – Protects data and manages multiple users.

Diagram how it works.



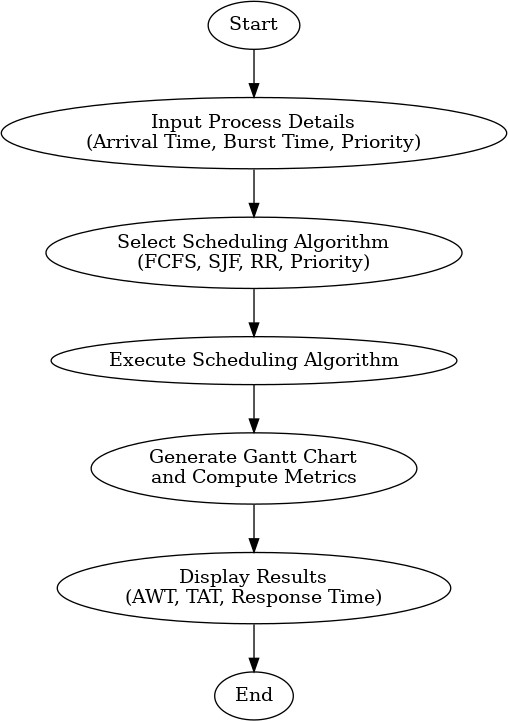
# Project Overview

The main goal of this project is to “develop a simulator that implements and compares different page replacement algorithms” used in operating systems. Page replacement algorithms are essential for managing virtual memory efficiently, ensuring that active processes get the necessary memory space without causing excessive delays. This simulator will help users understand how different algorithms handle page faults and optimize memory usage.  **Expected Outcomes**

1. **Implementation of Key Algorithms** o FIFO (First-In-First-Out) o LRU (Least Recently Used) o Optimal Page Replacement o LFU (Least Frequently Used)
2. **Real-Time Visualization** o Display the page replacement process step-by-step.

o Show how pages are allocated, replaced, and how page faults occur.

1. **Performance Metrics Calculation** o Page Fault Rate o Hit Ratio o Execution Time
2. **Comparison & Analysis** o Evaluate the efficiency of different page replacement algorithms in various scenarios. o Help users choose the best algorithm for a given situation.



 **Module Wise Breakdown**

**1️. Algorithm Execution Module (Core Logic) Purpose:**

* Implements various **page replacement algorithms** like FIFO, LRU, Optimal, and LFU.
* Handles memory allocation and **page fault handling** based on the selected algorithm.
* Keeps track of **page hits, faults, and memory states** throughout execution.

**Role in the Project:**

Simulates how pages are loaded and replaced in memory.

Processes user input (e.g., number of pages, page references).

Computes **performance metrics** (hit ratio, page fault rate, execution time).

**2️**. **Graphical User Interface (GUI) Module**   **Purpose:**

* Provides a **user-friendly interface** to interact with the simulator.
* Allows users to select different **page replacement algorithms** for comparison.
* Takes user input for **process details** (number of frames, page reference string, etc.).

**Role in the Project:**

Displays step-by-step execution of page replacement.

Provides an **interactive experience** for users.

Shows **Gantt charts or memory tables** to visualize the process. **Tech Stack:** Java (JavaFX), or Web-based (HTML, CSS, JavaScript).

**3️**. **Data Visualization & Analysis Module Purpose:**

* Analyzes the **performance of different algorithms** in various scenarios.
* Helps users **identify the most efficient algorithm** based on the results.

## Functionalities

**Module 1: Algorithm Execution Module Key Features:**

* **Algorithm Implementation:**
  + - Implements page replacement algorithms like **FIFO, LRU, and Optimal**.
    - **Example:** For FIFO, the module uses a queue to manage frames and determines which page to remove when a new page is needed.
* **Input Processing & Validation:**
  + - Accepts user input for the page reference string and number of frames.
    - Checks for valid input and handles errors gracefully.
    - **Example:** Ensures that the number of pages and frames are positive integers and that the page reference string contains valid numbers.
* **Performance Calculation:**

o Computes essential performance metrics such as **page fault count**, **hit ratio**, and **execution time**. o **Example:** After running the LRU algorithm, it displays “Total Page Faults: 8” and “Hit Ratio: 40%.”

* **Step-by-Step Simulation:**
  + - Provides detailed, step-by-step updates on the state of memory frames as pages are loaded and replaced.
    - **Example:** After each page request, the module prints the current frame configuration (e.g., [7, 0, 1]).

**Module 2: User Interface (CLI) Module Key Features:**

* **Interactive Command-Line Input:**

oPrompts users to input required values such as the number of frames, page reference string, and algorithm choice. o **Example:** Displays “Enter the number of frames:” and then waits for user input.

* **Clear and Organized Output:**
  + - Formats and displays the simulation progress and results in a user-friendly manner.
    - **Example:** Prints a formatted table or list that shows the frame state after each operation.
* **Algorithm Selection Menu:**

oOffers a simple menu to choose among implemented algorithms. o **Example:** Displays “1. FIFO, 2. LRU, 3. Optimal” for the user to select the desired algorithm.

* **Re-run Option:**

oProvides an option to restart the simulation with different parameters without exiting the program. o **Example:** After simulation completion, the CLI may ask “Do you want to run another simulation? (y/n)

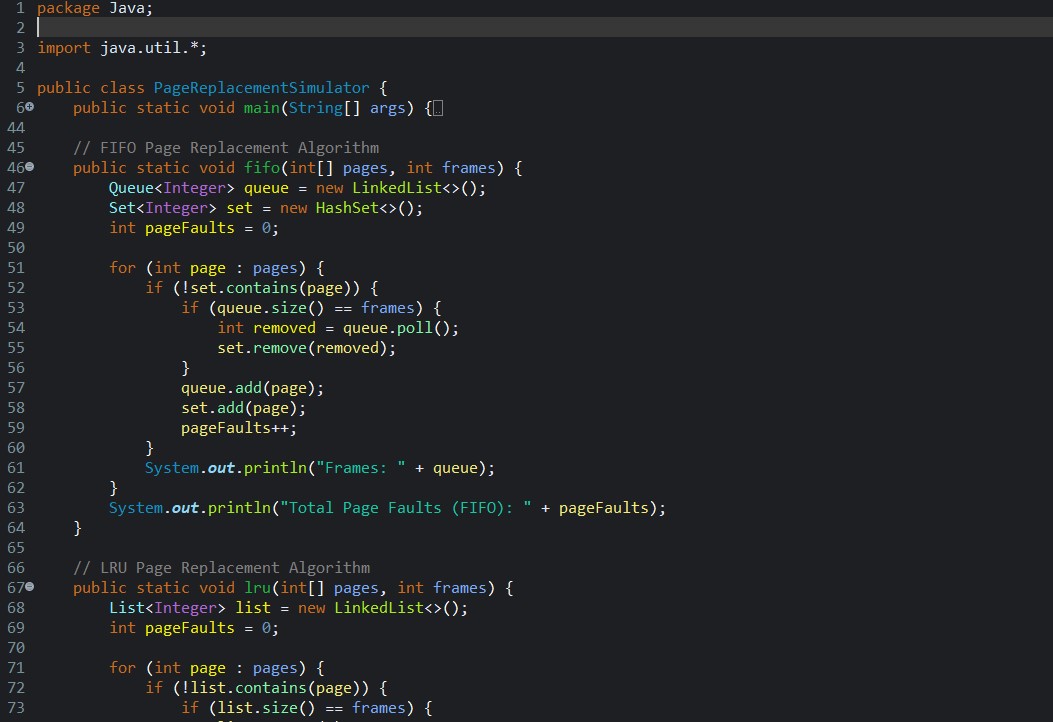


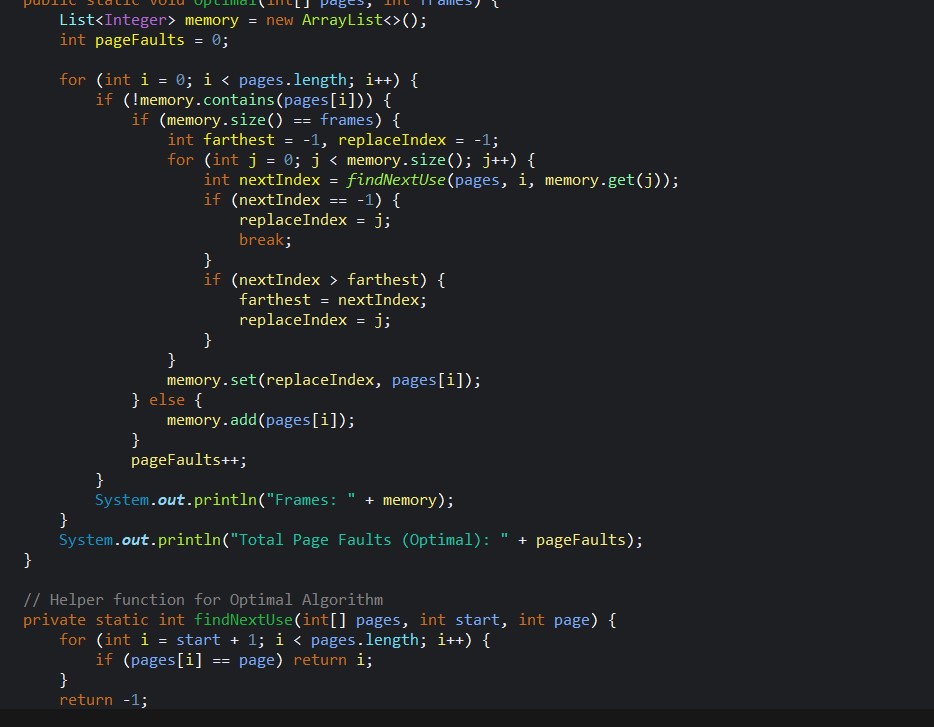
E

xecution and Output

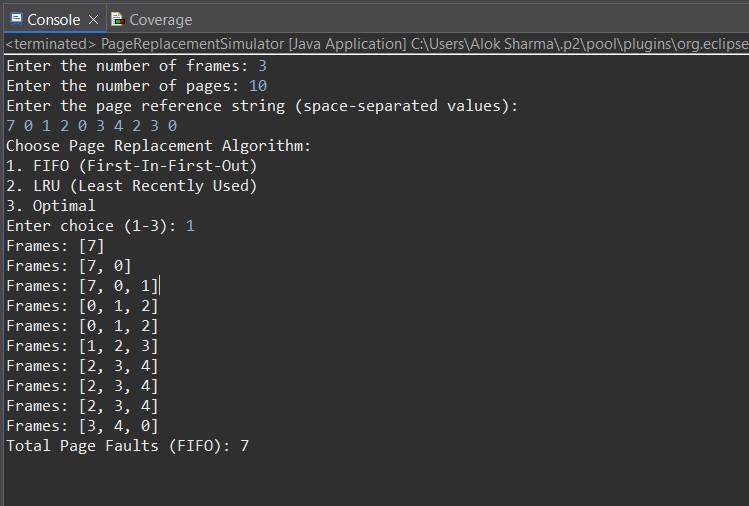
:

-





Output



* + Technology Recommendations.

Below are some technology recommendations for your **Efficient Page Replacement Algorithm Simulator** project:

**Programming Languages**

* **Java:**
  + - **Pros:** Robust, widely used for desktop applications, strong object-oriented support, and crossplatform compatibility.
    - **Usage:** Ideal if you’re planning to build a CLI and later transition to a GUI (using Swing or JavaFX).
* **Python:**
  + - **Pros:** Quick to develop, extensive libraries for data visualization and analysis, and a gentle learning curve.
    - **Usage:** Suitable if you want rapid prototyping, especially for visualizing performance metrics using libraries like Matplotlib.
* **C++:** o **Pros:** Excellent performance and low-level memory management control.

o **Usage:** Good if performance is a priority, though it might be more complex for visualization compared to Java or Python.

**Libraries & Frameworks**

* **For Java:** o **JavaFX or Swing:** For building a graphical user interface if you later decide to add a GUI.

o **JFreeChart:** A popular library for creating charts and graphs, useful for visualizing performance metrics (page faults, hit ratios, etc.).

* **For Python:**
  + - **Tkinter or PyQt:** For creating desktop GUI applications. o **Matplotlib or Seaborn:** For plotting graphs and visualizing data such as algorithm performance comparisons.
    - **Jupyter Notebook:** For interactive exploration and demonstration of simulation results.
* **For C++ (if chosen):** o **Qt:** For developing cross-platform GUI applications that include robust visualization options. o **SFML or SDL:** For simpler visualizations and graphical representations, if needed.

**Tools & Development Environment**

* **IDE:**
  + - **Java:** IntelliJ IDEA or Eclipse can help manage your project and provide robust debugging tools.
    - **Python:** PyCharm or VSCode for Python development offer integrated support for visualization libraries.
* **Version Control:** o **Git:** Use Git for version control and GitHub for hosting your repository. This will help you track changes and collaborate if needed.
* **Build Tools:**
  + - **Maven/Gradle (Java):** For managing project dependencies and automating builds.
    - **pip & virtualenv (Python):** For managing packages and creating isolated development environments.

**Why These Recommendations?**

* **Flexibility:** Java and Python both allow you to start with a CLI and later extend to a GUI, ensuring that your simulator is both interactive and visually appealing.
* **Visualization:** Tools like JFreeChart (Java) or Matplotlib (Python) provide robust solutions for plotting performance metrics, which is key to understanding algorithm efficiency.
* **Community Support:** Both languages have vast communities and abundant online resources to help you troubleshoot and optimize your project.
  + Execution Plan:

Execution Plan for Efficient Page Replacement Algorithm Simulator

This plan outlines a **step-by-step approach** to developing your simulator, ensuring smooth progress and structured implementation.

**Phase 1: Planning & Setup**

**Step 1: Define Requirements**

Decide the algorithms to be implemented (**FIFO, LRU, Optimal, etc.**).

Determine input format (page references, frames, etc.).

Define output metrics (page faults, hit ratio, execution time).

Choose visualization requirements (graphs, text-based simulation, GUI in the future). **Step 2: Set Up Development Environment**

* Install Java (JDK 17+).
* Choose an **IDE (IntelliJ, Eclipse, VS Code, etc.)**.
* Set up **GitHub repository** for version control.
* Install dependencies (e.g., JFreeChart if visualization is required).

**Phase 2: Core Implementation (CLI Version First)**  **Step 3: Implement Page Replacement Algorithms** Each algorithm will have:

* **Input:** Number of frames, sequence of page requests.
* **Processing:** Simulate how pages are replaced in memory.
* **Output:** Total page faults, hit ratio, and execution logs.

**Subtasks:**

* Implement **FIFO (First-In-First-Out)**.
* Implement **LRU (Least Recently Used)**.
* Implement **Optimal Page Replacement**.
* Test each algorithm separately.

**Step 4: Implement Performance Metrics Calculation**

* Compute **Page Fault Rate** & **Hit Ratio**.
* Track **Memory Usage & Execution Time**.
* Store results for **comparative analysis**.

**Step 5: Command-Line Interface (CLI) for User Input**

* Allow users to **select an algorithm**.
* Take **user-defined page sequences** or **generate random sequences**.
* Display results **clearly** (tabular format).

**Phase 3: Advanced Features & Visualization**  **Step 6: Implement Real-Time Simulation (Optional)**

* Show how pages are replaced **step by step**.
* Animate memory frame updates in CLI (simple ASCII representation). **Step 7: Add Visualization (Optional - GUI or Charts)**
* Use **JFreeChart (Java)** to create graphs comparing page faults of different algorithms.
* Provide **graphical representation of memory state over time**. **Step 8: Add Data Export Feature**
* Save results as **CSV or JSON** for later analysis.

**Phase 4: Testing & Deployment**

**Step 9: Unit Testing & Debugging**

* Test each algorithm with **various test cases**.
* Compare outputs with expected results from theoretical examples. **Step 10: Optimize Performance & Refactor Code**
* Reduce redundant computations.
* Improve code readability and modularity.

**Step 11: Publish on GitHub & Documentation**

* Write a **README.md** explaining usage, setup, and examples.
* Include sample inputs and expected outputs.
* Optionally, record a **demo video** explaining the project.

## Revision Tracking on GitHub

• **Repository Name:** Efficient Page Replacement Algorithm Simulator

**GitHub Link:-** [https://github.com/alokkksharmaa/Efficient-Page-Replacement-AlgorithmSimulator/tree/main](https://github.com/alokkksharmaa/Efficient-Page-Replacement-Algorithm-Simulator/tree/main)