**Page Replacement Algorithm Simulator**

**Introduction:**

This program simulates three-page **replacement algorithms**:

1. **FIFO (First-In-First-Out)**
2. **LRU (Least Recently Used)**
3. **Optimal**

It allows the user to:

* Input a **page reference string** (a sequence of page requests).
* Select an **algorithm** from FIFO, LRU, or Optimal.
* Set the **number of frames** (memory slots available for pages).
* View a **graphical representation** of how the pages are stored in memory over time.
* Get statistics on **page faults and hit ratio**.

Importing Required Libraries

1. import tkinter as tk

2. from tkinter import ttk, messagebox

3. import numpy as np

4. import matplotlib.pyplot as plt

* tkinter: Python’s built-in library for creating GUI applications.
* ttk: Provides themed widgets (e.g., dropdown menus, buttons) for better UI.
* messagebox: Used to show popup messages (e.g., errors, results).
* numpy (np): Used for handling arrays and numerical data.
* matplotlib.pyplot (plt): Used to **plot** the page replacement process.

FIFO Page Replacement Algorithm

1.def lru\_page\_replacement(pages, frames):

2. frame\_list = []

3. history = []

4. hits, misses = 0, 0

5. page\_order = [] # Tracks order of pages for LRU

6.

7. for page in pages:

8. if page in frame\_list:

9. hits += 1

10. page\_order.remove(page) # Update usage order

11. else:

12. misses += 1

13. if len(frame\_list) < frames:

14. frame\_list.append(page)

15. else:

16. lru\_page = page\_order.pop(0) # Remove least recently used page

17. frame\_list[frame\_list.index(lru\_page)] = page

18.

19. page\_order.append(page) # Update order for next use

20. history.append(frame\_list.copy())

21.

22. return history, hits, misses

23.

* **LRU removes the least recently used page**.
* Uses page\_order to **track page usage**.
* When memory is full:
  + It removes the **page that was used the longest time ago**.
  + Updates page\_order every time a page is accessed.

Optimal Page Replacement Algorithm

1. def optimal\_page\_replacement(pages, frames):

2. frame\_list = []

3. history = []

4. hits, misses = 0, 0

5.

6. for i in range(len(pages)):

7. page = pages[i]

8.

9. if page in frame\_list:

10. hits += 1

11. else:

12. misses += 1

13. if len(frame\_list) < frames:

14. frame\_list.append(page)

15. else:

16. future\_use = {frame: pages[i+1:].index(frame) if frame in pages[i+1:] else float('inf') for frame in frame\_list}

17. page\_to\_replace = max(future\_use, key=future\_use.get)

18. frame\_list[frame\_list.index(page\_to\_replace)] = page

19.

20. history.append(frame\_list.copy())

21.

22. return history, hits, misses

* **Optimal removes the page that will be used farthest in the future**.
* It scans ahead to determine **which page will be needed last**.
* This results in the **fewest page faults** but is impractical in real-world systems.