CS 4310 Operating Systems
Project #2 Simulating Page Replacement Manager and Performance Analysis

Due: 12/3
(Total: 100 points)

Student Name: Loc Nguyen

Date: 12/1/2024

Important:
- Please read this document completely before you start coding.
- Also, please read the submission instructions (provided at the end of this document) carefully before submitting the project.

Project #2 Description:

Simulating Page Replacement Manager of the Operating Systems by programming the following three page replacement algorithms that we covered in the class:

a. First In First Out (FIFO)
b. Least Recently Used (LRU)
c. Optimal Algorithm (OPT)

You can use either Java, or C++ for the implementation. The objective of this project is to help student understand how above three page replacement algorithms operates by implementing the algorithms, and conducting a performance analysis of them based on the performance measure of ***page faults*** for each page replacement algorithm using multiple inputs. Output the details of each algorithm's execution. You need to show what pages are inside the page frames along with the reference string and mark it when a page fault occurred. You can choose your display format, for examples, you can display the results for each reference string in a table format as shown in the class notes. The project will be divided into three phases to help you to accomplish above tasks in in a systematic and scientific fashion: Design and Testing, Implementation, and Performance Analysis.

The program will read in a testing data file consisting 50 reference strings of length 30 (TestingData.txt) – this file will be generated by you. In this project, assume that

(1) the length of the reference string is always 30, e.g. 361724720354720146353214567012.
(2) there are 8 pages, from 0 to 7.

You are required to run this testing data file using the three page replacement algorithms with four different page frame sizes (3, 4, 5 and 6).

You can structure the testing data in your own way.
A look of a sample testing data file, named "TestingData.txt", is given as follows:

```
[Begin of TestingData.txt]
361724720354720146353214567012
726301625203767261726562516250
…
342514361524372635435261547265
[End of TestingData.txt]
```

You can implement the algorithms in your choice of data structures based on the program language of your choice. Note that you always try your best to give the most efficient program for each problem.

Submission Instructions:

- ***turn in the following @canvas.cpp.edu after the completion of all three parts, part 1, part 2 and part 3***
    - ***(1) testing data file (with 50 reference strings of length 30)***
    - ***(2) three program files (your choice of programming language with proper documentation)***
    - ***(3) this document (complete all the answers)***

Part 1
Design & Testing (30 points)

a. Design the program by providing pseudocode or flowchart for each page replacement algorithm.

```
FIFO
Current frame = []
Page_fault_counter = 0
Result_frame = []

For page in reference_str:
    If page not in current frame:
            If length of current frame is >= frame_size
                    Remove the oldest page
            Append the current page to the frame
            Add 1 to page_fault_counter
    Append current frame to list of result_frame

Return result_frame and page_fault_counter
```

```
LRU
Current frame = []
Page_fault_counter = 0
Result_frame = []

For page in reference_str:
    If page not in current frame:
            If length of current frame is >= frame_size
                    Remove the oldest page
            Append the current page to the frame
            Add 1 to page_fault_counter

    Else:
            Remove the current page
            Add it to the end of the frame                    #to indicate recency
    Append current frame to list of result_frame

Return result_frame and page_fault_counter
```

```
Optimal
Current frame = []
Page_fault_counter = 0
Result_frame = []

For page in reference_str:
    If page not in current frame:
            If length of current frame is >= frame_size
                    Look ahead in string to see when it will be used next
                    If it’s not used again, assume the next use is at infinity
                    Keep track of page that will not be used for the longest time
            Remove farthest page from frame
            Append the new page to frames

    Return frame_result, page_fault_counter
```

b. Design the program correctness testing cases. Give at least 4 testing cases (with 3, 4, 5, or 6, page frames) to test your program, and give the expected correct output (# of page faults) of the program for each case in order to test the correctness of each algorithm.

<complete the following table>

Testing case #	Input reference string	Expected # of page faults for FIFO (√ if Correct after testing in Part 3)	Expected # of page faults for LRU (√ if Correct after testing in Part 3)	Expected # of page faults for Optimal Algorithm (√ if Correct after testing in Part 3)
1 (3 page frames)	565371052760413265407623517204	29√	29√	20√
2 (4 page frames)	203715467530624150764052367214	26√	27√	18√
3 (5 page frames)	040537261740653025712614735614	22√	24√	15√
4 (6 page frames)	135627401560743152067215347621	16√	23√	12√

c. Design testing strategy for the programs. Discuss about how to generate and structure the randomly generated inputs for experimental study later in Part 3.

To study the performance of the three page replacement algorithms, let’s use a random number generator for generating the testing data set file of 50 reference strings of length 30 as the input for the programs. However, student should use this same data set for running each of the three page replacement algorithms and each of the four frame sizes (3, 4, 5, 6).

The average performance (average number of page faults) can be calculated after an experiment is conducted using the testing data set file of 50 reference strings of length 30.

For example, if $X = \{x_1, x_2, x_3 \ldots x_{50}\}$ *contains 50 results (number of page faults) using the testing data set as input for FIFO and page frame size of 3, then*

average number of page faults for FIFO and page frame size of 3 $= \frac{\sum_{i=1}^{50} xi}{50}$

Calculate average page faults for each reference string for each size of frame and average them out at the end to compare.

Part 2
Implementation (30 points)

a. Code each program based on the design (pseudocode or flow chart) in Part 1(a).

   Done!

b. Document the program appropriately.

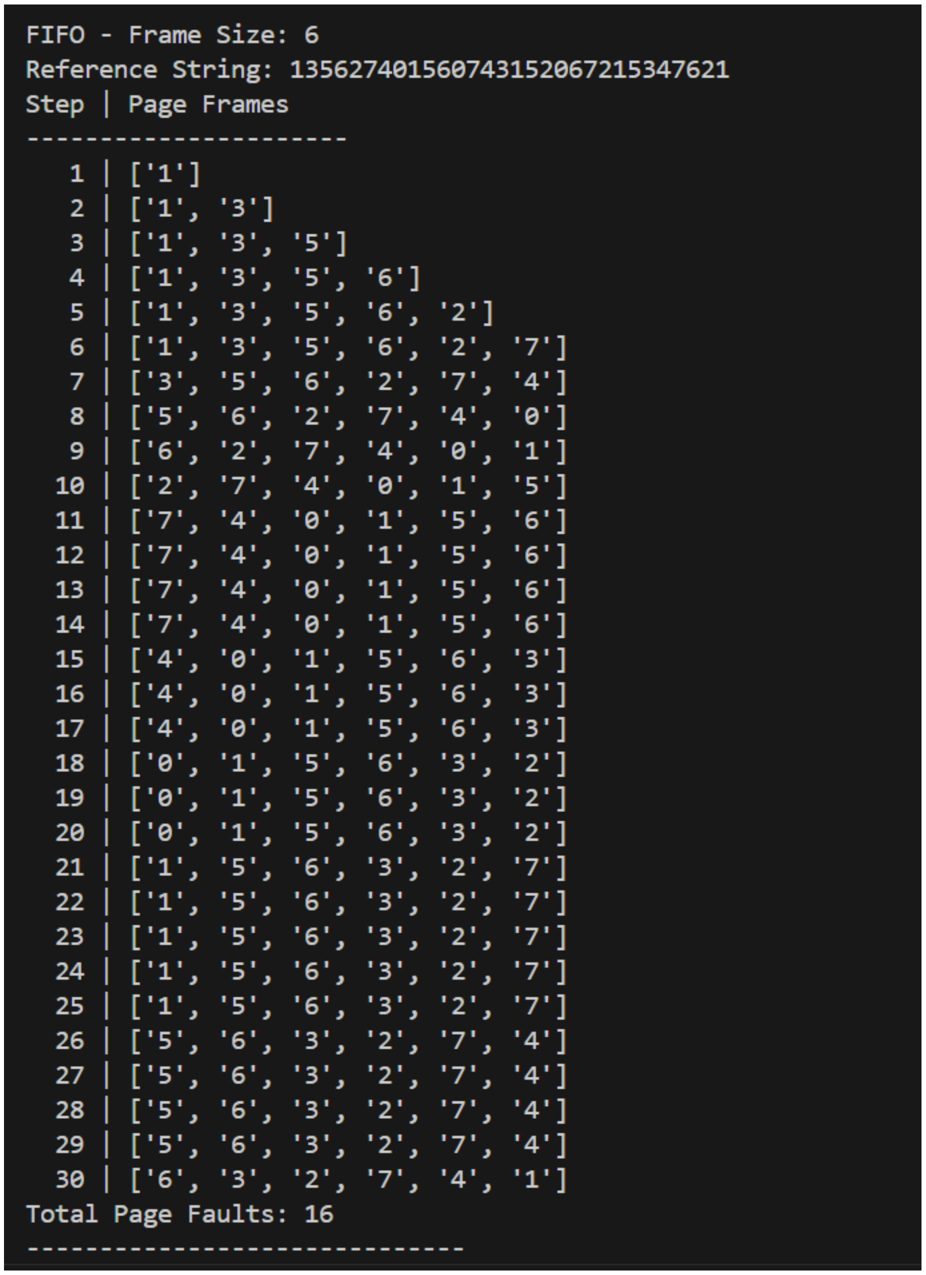
   Done!

c. Test you program using the designed testing input data given in the table in Part 1(b), Make sure each program generates the correct answer by marking a "√" if it is correct for each testing case for each program column in the table. Repeat the process of debugging if necessary.

   Done!

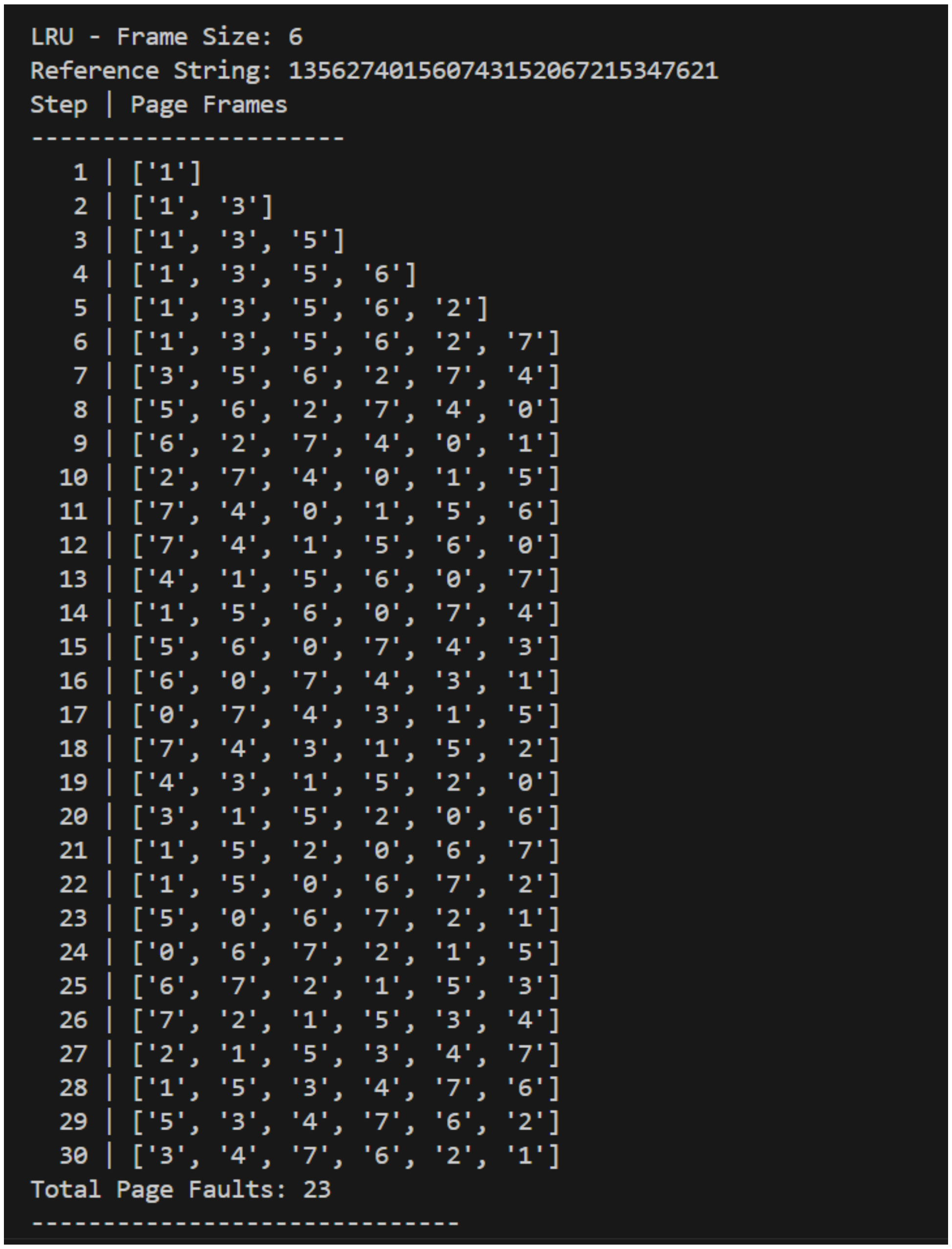
d. For each page replacement program, capture a screen shot of the execution (Compile&Run) of one testing reference string to show how this program works properly

FIFO



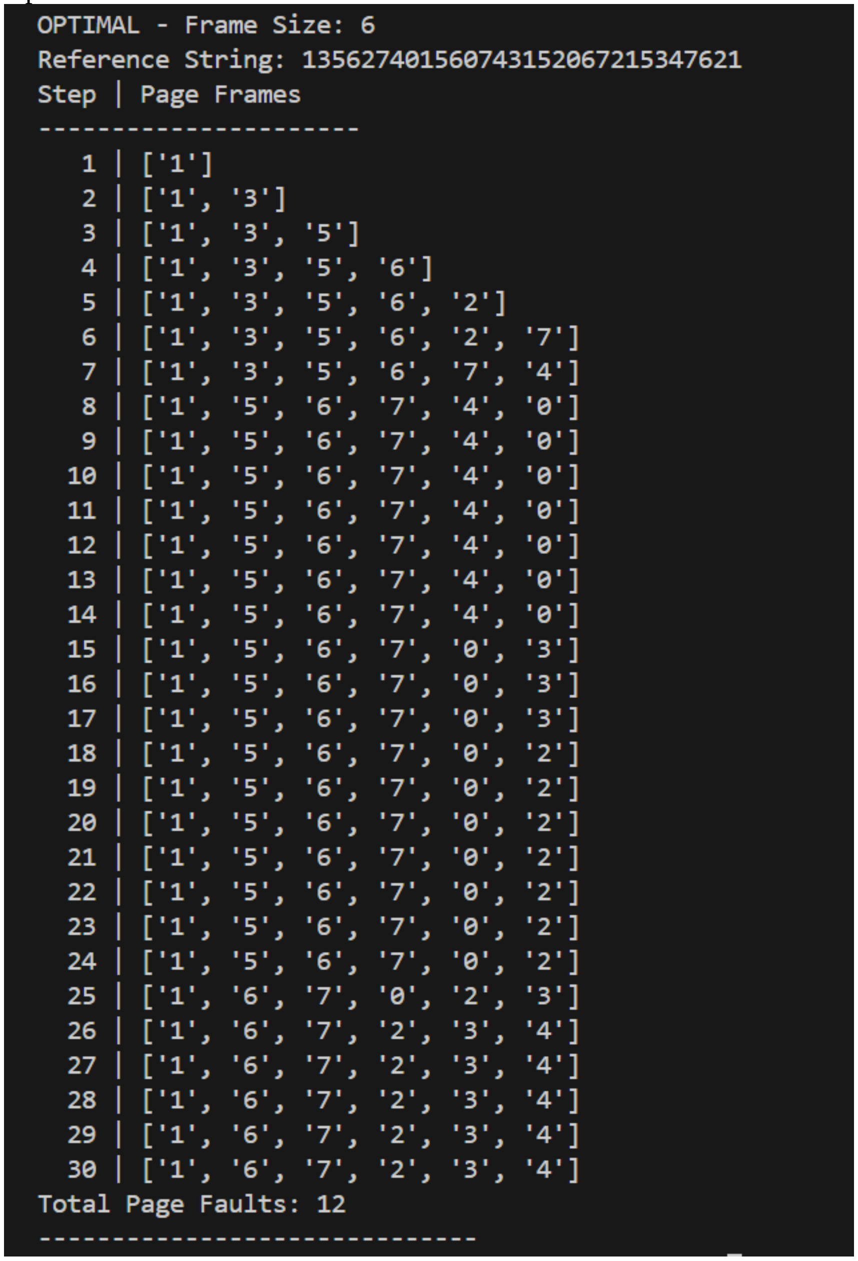
```
FIFO - Frame Size: 6
Reference String: 135627401560743152067215347621
Step | Page Frames
----------------------
   1 | ['1']
   2 | ['1', '3']
   3 | ['1', '3', '5']
   4 | ['1', '3', '5', '6']
   5 | ['1', '3', '5', '6', '2']
   6 | ['1', '3', '5', '6', '2', '7']
   7 | ['3', '5', '6', '2', '7', '4']
   8 | ['5', '6', '2', '7', '4', '0']
   9 | ['6', '2', '7', '4', '0', '1']
  10 | ['2', '7', '4', '0', '1', '5']
  11 | ['7', '4', '0', '1', '5', '6']
  12 | ['7', '4', '0', '1', '5', '6']
  13 | ['7', '4', '0', '1', '5', '6']
  14 | ['7', '4', '0', '1', '5', '6']
  15 | ['4', '0', '1', '5', '6', '3']
  16 | ['4', '0', '1', '5', '6', '3']
  17 | ['4', '0', '1', '5', '6', '3']
  18 | ['0', '1', '5', '6', '3', '2']
  19 | ['0', '1', '5', '6', '3', '2']
  20 | ['0', '1', '5', '6', '3', '2']
  21 | ['1', '5', '6', '3', '2', '7']
  22 | ['1', '5', '6', '3', '2', '7']
  23 | ['1', '5', '6', '3', '2', '7']
  24 | ['1', '5', '6', '3', '2', '7']
  25 | ['1', '5', '6', '3', '2', '7']
  26 | ['5', '6', '3', '2', '7', '4']
  27 | ['5', '6', '3', '2', '7', '4']
  28 | ['5', '6', '3', '2', '7', '4']
  29 | ['5', '6', '3', '2', '7', '4']
  30 | ['6', '3', '2', '7', '4', '1']
Total Page Faults: 16
------------------------------
```

LRU



```
LRU - Frame Size: 6
Reference String: 135627401560743152067215347621
Step | Page Frames
----------------------
   1 | ['1']
   2 | ['1', '3']
   3 | ['1', '3', '5']
   4 | ['1', '3', '5', '6']
   5 | ['1', '3', '5', '6', '2']
   6 | ['1', '3', '5', '6', '2', '7']
   7 | ['3', '5', '6', '2', '7', '4']
   8 | ['5', '6', '2', '7', '4', '0']
   9 | ['6', '2', '7', '4', '0', '1']
  10 | ['2', '7', '4', '0', '1', '5']
  11 | ['7', '4', '0', '1', '5', '6']
  12 | ['7', '4', '1', '5', '6', '0']
  13 | ['4', '1', '5', '6', '0', '7']
  14 | ['1', '5', '6', '0', '7', '4']
  15 | ['5', '6', '0', '7', '4', '3']
  16 | ['6', '0', '7', '4', '3', '1']
  17 | ['0', '7', '4', '3', '1', '5']
  18 | ['7', '4', '3', '1', '5', '2']
  19 | ['4', '3', '1', '5', '2', '0']
  20 | ['3', '1', '5', '2', '0', '6']
  21 | ['1', '5', '2', '0', '6', '7']
  22 | ['1', '5', '0', '6', '7', '2']
  23 | ['5', '0', '6', '7', '2', '1']
  24 | ['0', '6', '7', '2', '1', '5']
  25 | ['6', '7', '2', '1', '5', '3']
  26 | ['7', '2', '1', '5', '3', '4']
  27 | ['2', '1', '5', '3', '4', '7']
  28 | ['1', '5', '3', '4', '7', '6']
  29 | ['5', '3', '4', '7', '6', '2']
  30 | ['3', '4', '7', '6', '2', '1']
Total Page Faults: 23
------------------------------
```

Optimal



```
OPTIMAL - Frame Size: 6
Reference String: 135627401560743152067215347621
Step | Page Frames
----------------------
   1 | ['1']
   2 | ['1', '3']
   3 | ['1', '3', '5']
   4 | ['1', '3', '5', '6']
   5 | ['1', '3', '5', '6', '2']
   6 | ['1', '3', '5', '6', '2', '7']
   7 | ['1', '3', '5', '6', '7', '4']
   8 | ['1', '5', '6', '7', '4', '0']
   9 | ['1', '5', '6', '7', '4', '0']
  10 | ['1', '5', '6', '7', '4', '0']
  11 | ['1', '5', '6', '7', '4', '0']
  12 | ['1', '5', '6', '7', '4', '0']
  13 | ['1', '5', '6', '7', '4', '0']
  14 | ['1', '5', '6', '7', '4', '0']
  15 | ['1', '5', '6', '7', '0', '3']
  16 | ['1', '5', '6', '7', '0', '3']
  17 | ['1', '5', '6', '7', '0', '3']
  18 | ['1', '5', '6', '7', '0', '2']
  19 | ['1', '5', '6', '7', '0', '2']
  20 | ['1', '5', '6', '7', '0', '2']
  21 | ['1', '5', '6', '7', '0', '2']
  22 | ['1', '5', '6', '7', '0', '2']
  23 | ['1', '5', '6', '7', '0', '2']
  24 | ['1', '5', '6', '7', '0', '2']
  25 | ['1', '6', '7', '0', '2', '3']
  26 | ['1', '6', '7', '2', '3', '4']
  27 | ['1', '6', '7', '2', '3', '4']
  28 | ['1', '6', '7', '2', '3', '4']
  29 | ['1', '6', '7', '2', '3', '4']
  30 | ['1', '6', '7', '2', '3', '4']
Total Page Faults: 12
------------------------------
```

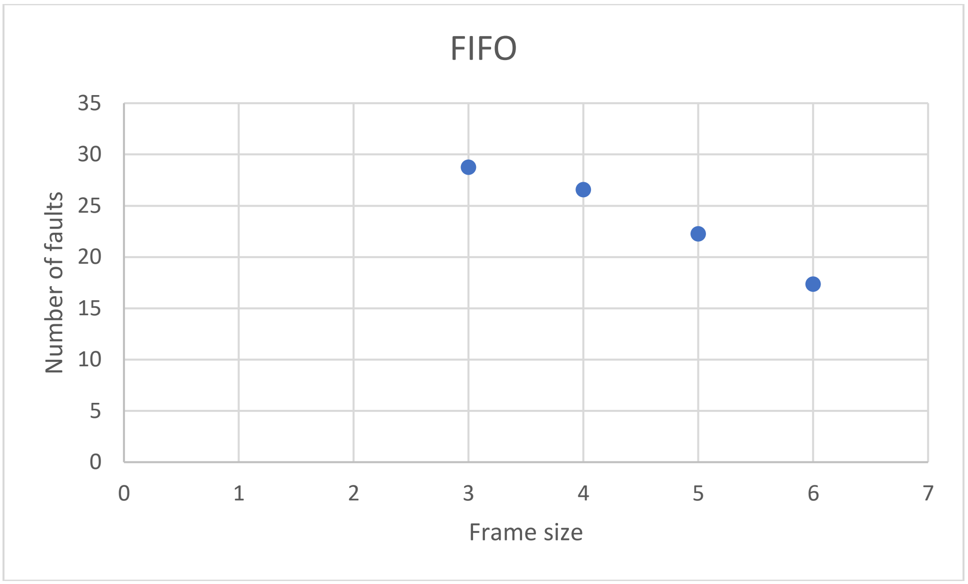
By now, three working programs are created and ready for experimental study in the next part, Part 3.

Part 3
Performance Analysis (40 points)

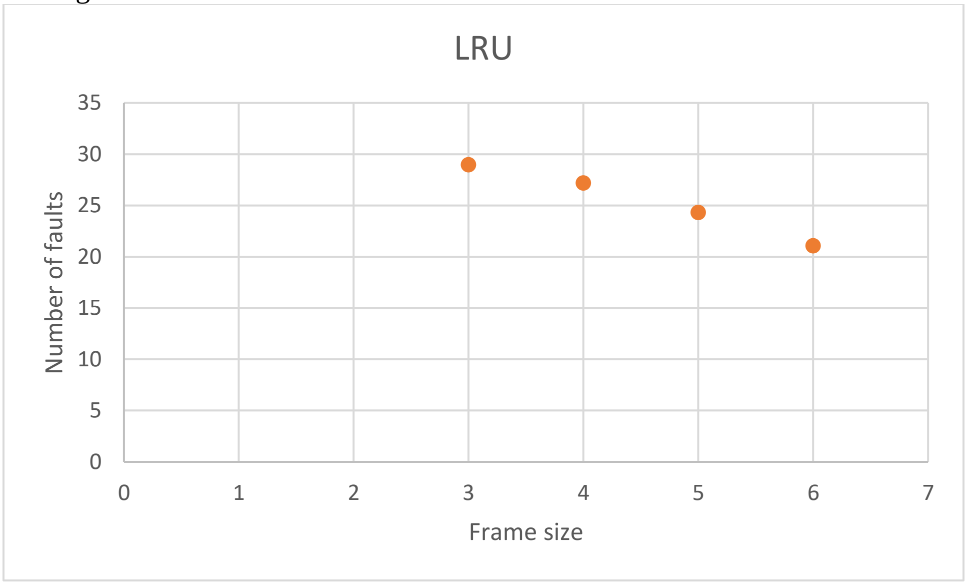
a. Run each program with the designed randomly generated input data given in Part 1(c). Generate a table for all the experimental results for performance analysis as follows.

number of page frames	Average number of page faults (FIFO Program)	Average number of page faults (LRU Program)	Average number of page faults (Optimal Algorithm)
3 page frames	28.78	28.96	21.22
4 page frames	26.58	27.2	17.94
5 page frames	22.26	24.3	14.94
6 page frames	17.38	21.06	12.18

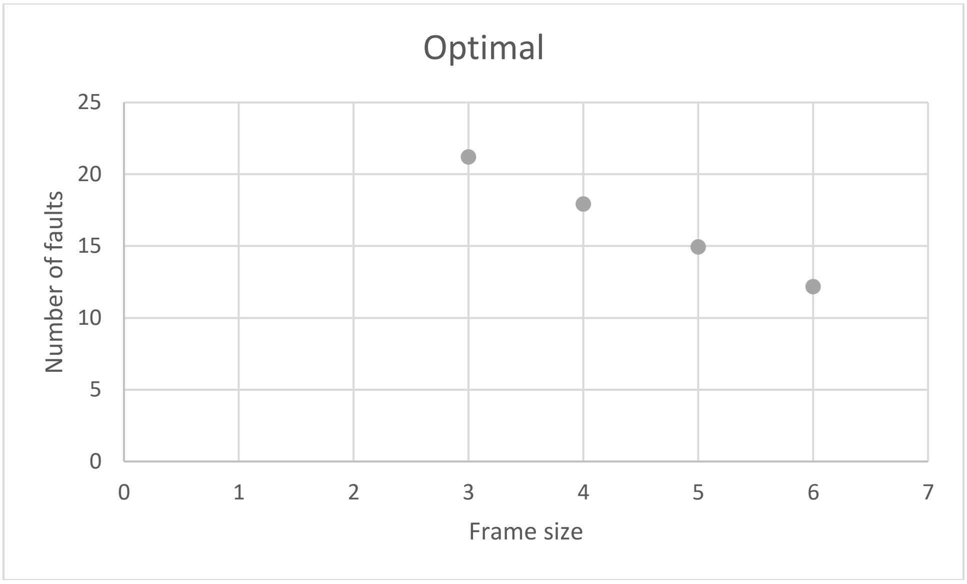
b. Plot a graph of each algorithm, average page fault vs. page frame size (3, 4, 5, 6) and summarize the performance of each algorithm based on its own graph.



Average number of faults decrease as frame size increases

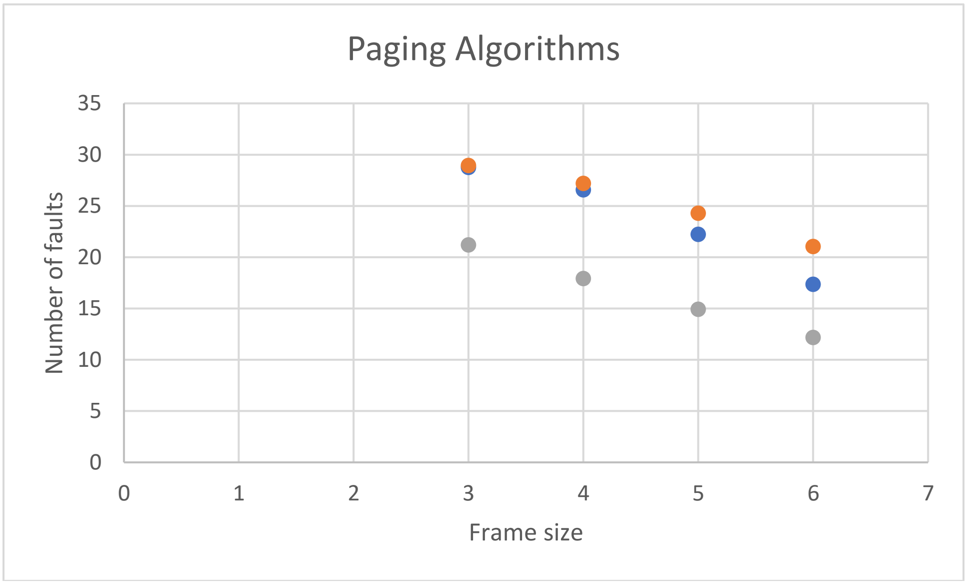


The average number of faults decrease as frame size increases but less dramatic versus FIFO



The average number of faults decrease as frame size increases almost linearly

Plot all three graphs on the same graph and compare the performance (page faults) of all three algorithms. Rank three page replacement algorithms. Try giving the reasons for the findings.



FIFO and LRU are similar but on average, FIFO performs better than LRU due to having the property of having sets of working pages fit within the time frame. However, FIFO can experience Belady’s Anomaly in some cases as frame size increases. Optimal performs the best since it uses look-ahead to determine the best course of action to retain or remove a page in the frame.

c. Conclude your report with the strength and constraints of your work. At least 100 words. (Note: It is reflection of this project. If you have a change to re-do this project again, what you like to keep and what you like to do differently in order get a better quality of results.)

   Using the datasets I've generated from consisting of pages from 0 to 7, it's completely random. In reality, pages may have a certain order and therefore, isn't reflected well within this simulation. On another note, Belady Anomaly's cases didn't appear as often and isn't considered more seriously in this case. I need a larger sample size than just 50 strings and possibly increasing the frame size as well as changing the substance of the pages numbering.
   When designing the experiment, I ran into some trouble trying to implement the data structure in such a way that it doesn't affect the result stemming from the algorithm itself. It took a while to implement but that was an aspect of the project that I spend the most time on.