Page Replacement Algorithms in Operating Systems

In an operating system that uses paging for memory management, a page replacement algorithm is needed to decide which page needs to be replaced when new page comes in.

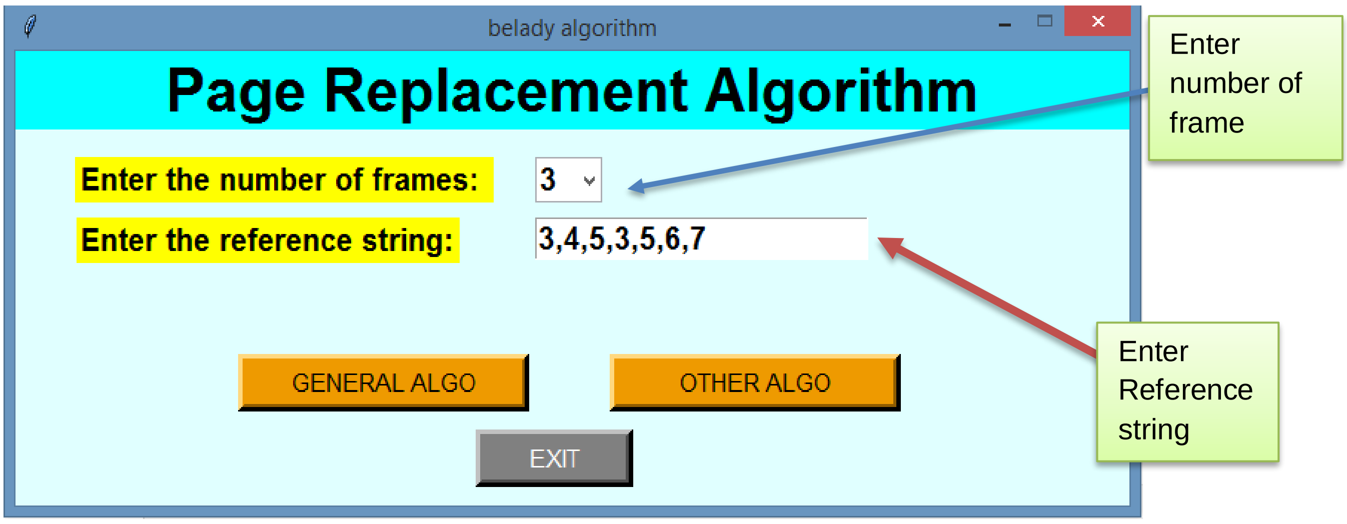
Page Fault – A page fault happens when a running program accesses a memory page that is mapped into the virtual address space, but not loaded in physical memory.Since actual physical memory is much smaller than virtual memory, page faults happen. In case of page fault, Operating System might have to replace one of the existing pages with the newly needed page. Different page replacement algorithms suggest different ways to decide which page to replace. The target for all algorithms is to reduce the number of page faults.

Types of Page Replacement Algorithms

There are various **page replacement techniques**, and we will explain each one

- **FIFO Page Replacement Algorithm**
- **LIFO Page Replacement Algorithm**
- **LRU Page Replacement Algorithm**
- **Optimal Page Replacement Algorithm**
- **Random Page Replacement Algorithm**

Given set of inputs for Page Replacement Algorithms.



FIFO Page Replacement Algorithm

This page replacement algorithm is very easy and simple because this algorithm is based on the "First in First out"principle. In which operating system uses the line as queue that store all information of all pages in the computer memory.
As per FIFO principle, oldest page is replaced at the front side and most recent page is replaced at the rear side.

FIFO (First In First Out)

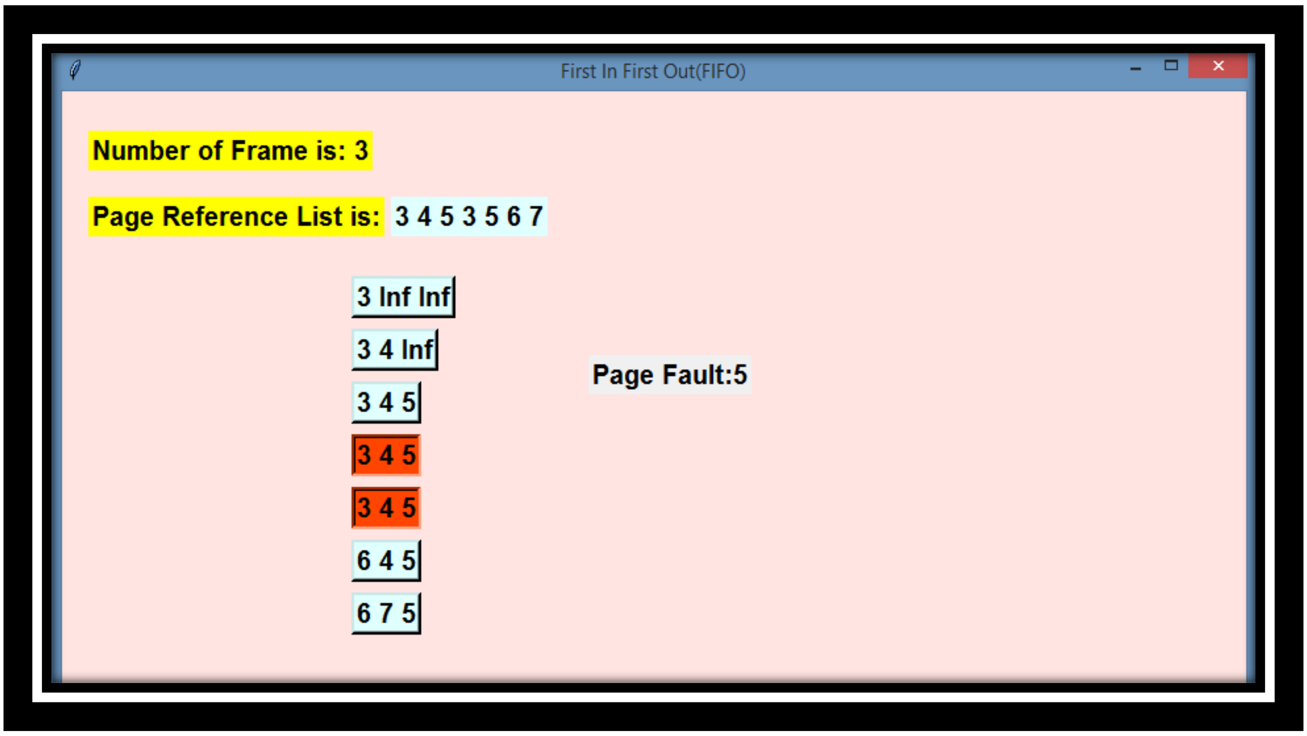
Method: Select the frame that has been in Main Memory the longest and remove it to make room.

Implementation: maintain a linked list data structure of all pages currently in Main Memory and remove the page that is pointed to by the head of the linked list.

CODE:

```
def FIFO1():
     Frame = [np.inf for i in range(NumberOfFrame)]
     print(Frame, '\t # Empty Frame')
     j = 0
     x = 0
     PageFault = 0
     for i in range(len(PageReference)):
        PreIndex = j

        if PageReference[i] not in Frame:
            PageFault += 1
            Frame[j] = PageReference[i]
            j = (j + 1) % NumberOfFrame
        if PreIndex == j:
            print(Frame, '\t No Page Fault (Hit)')
            mylabel3    =    Label(topw,    text=Frame,    bg="orange   red",
    font="comicsansms 15 bold",borderwidth=3,relief=SUNKEN)
            mylabel3.place(x=220, y=140+(40*x))
        else:
            print(Frame, '\t occur Page Fault (Miss)')
            mylabel3    =    Label(topw,    text=Frame,    bg="light   cyan",
    font="comicsansms 15 bold",borderwidth=3,relief=RAISED)
            mylabel3.place(x=220, y=140+(40*x))
        x+=1
    print('Number of Page Fault: ', PageFault)
```

RESULT:



First In First Out(FIFO)

Number of Frame is: 3

Page Reference List is: 3 4 5 3 5 6 7

```
3 Inf Inf
3 4 Inf
3 4 5
3 4 5
3 4 5
6 4 5
6 7 5
```

Page Fault:5

LRU Page Replacement Algorithm

LRU stands for “Least Recently Used”, and it helps to operating system for searching such page that is used over the short duration of time frame. This page replacement algorithm uses the counter along with even page, and that counter is known as aging registers.

LRU algorithm helps to select that page which is not needed for long life in to primary memory.

LRU (Least Recently Used)

Method: Select the frame that has been referenced the least recently of all frames in Main Memory.

Implementation 1: Maintain a linked list of all pages currently in Main Memory and each time a page is referenced (read or write), move the page to the tail of the linked list

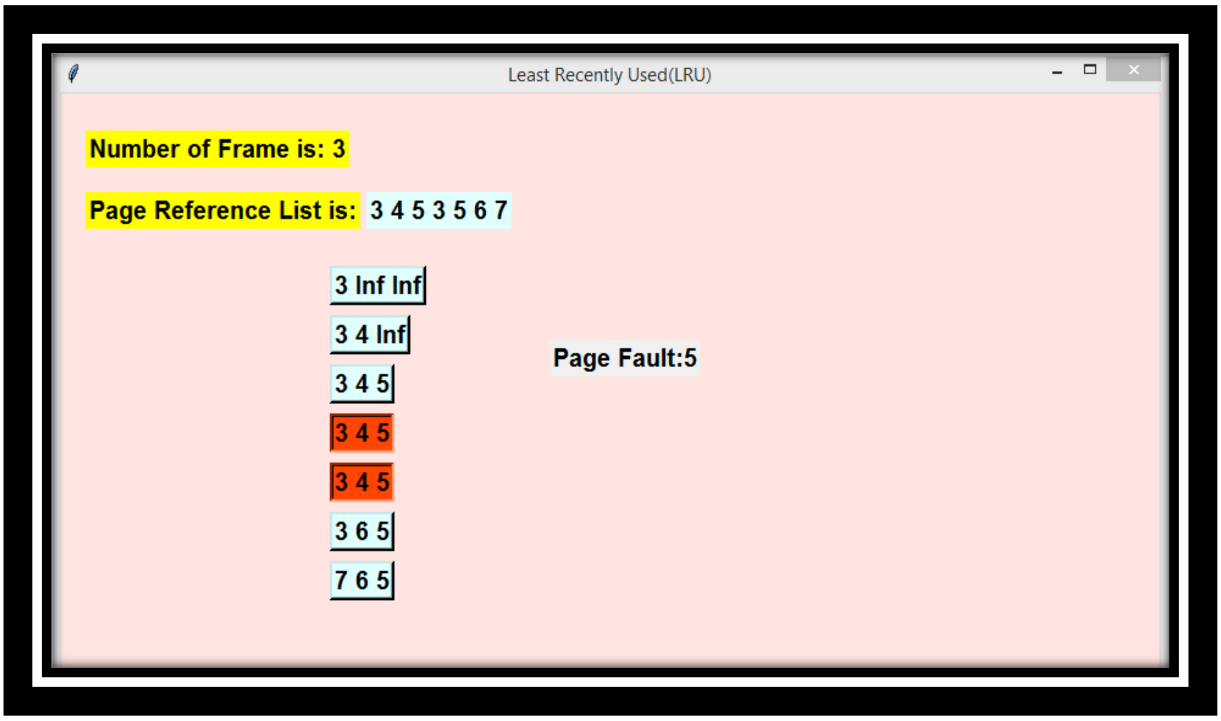
Implementation 2: Maintain a timestamps of all pages currently in Main Memory and each time a page is referenced (read or write), update the timestamp. When replacing a page, pick the page with the oldest timestamp.

CODE:

```
def LRU():
      Frame = [np.inf for i in range(NumberOfFrame)]
      IndexList = [i for i in range(NumberOfFrame)]
      print(Frame, '\t # Empty Frame')
      j = 0
      x=0
      PageFault = 0
      for i in range(len(PageReference)):
         if np.inf in Frame:
             index = Frame.index(np.inf)
             Frame[index] = PageReference[i]
             IndexList.pop(0)
             IndexList.append(index)
             PageFault += 1
             print(PageReference[i], Frame, '\t occur Page Fault (Miss)')
             mylabel3 = Label(topw, text=Frame, bg="light cyan", font="comicsansms 15
      bold", borderwidth=3, relief=RAISED)
             mylabel3.place(x=220, y=140+(40*x))
         elif PageReference[i] not in Frame:
             Pos = IndexList[0];
             Frame[Pos] = PageReference[i]
             IndexList.pop(0)
             IndexList.append(Pos)
             PageFault += 1
             print(PageReference[i], Frame, '\t occur Page Fault (Miss)')
```

```
            mylabel3 = Label(topw, text=Frame, bg="light cyan", font="comicsansms 15 bold", borderwidth=3, relief=RAISED)
            mylabel3.place(x=220, y=140+(40*x))
        else:
            index = Frame.index(PageReference[i])
            index_p = IndexList.index(index)
            IndexList.pop(index_p)
            IndexList.append(index)
            print(PageReference[i], Frame, '\t No Page Fault (Hit)')
            mylabel3 = Label(topw, text=Frame, bg="orange red", font="comicsansms 15 bold", borderwidth=3,relief=SUNKEN)
            mylabel3.place(x=220, y=140+(40*x))
        x+=1
    print('Number of Page Fault: ', PageFault)
```

RESULT:



Least Recently Used(LRU)

Number of Frame is: 3

Page Reference List is: 3 4 5 3 5 6 7

```
3 Inf Inf
3 4 Inf
3 4 5
3 4 5
3 4 5
3 6 5
7 6 5
```

Page Fault:5

Optimal Page Replacement Algorithm

Optimal Page Replacement Algorithm is very excellent page replacement policy because it helps to provide least number of page faults, so it is called of “OPT”, “Clairvoyant Replacement Algorithm”, and “Belady’s optimal page policy”.

Features are

- In this algorithm, such pages are replaced which are not needed to long life duration in future.
- This algorithm does not produce more page faults.
- It has very low page fault rate compare to other algorithm.

Optimal Page Replacement

Method: Look into the future at the pages that will be referenced(if possible) , and of all the pages currently in memory, select the page that’s next reference will be the furthest in the future.

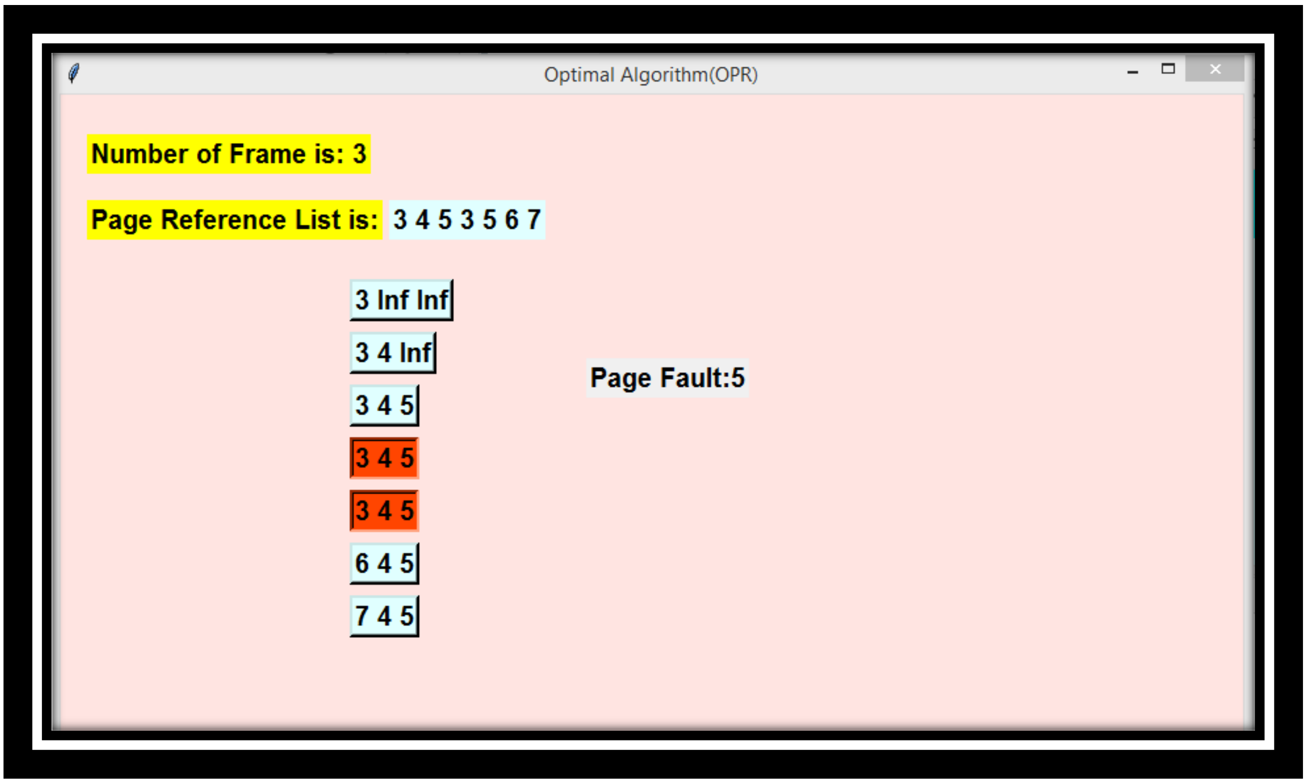
Implementation: Predict the future. The OS must know the exact sequence of referenced that the process will make. Not feasible if the program has if…then…else statements. Possible if the process is a straight process without decisions.

Note: Although this is not realistic to implement, it is the benchmark to check other algorithms against. That is, if the optimal algorithm produces x page faults, you would compare other algorithms against x.

CODE:

```
def OPR():

        Frame = [np.inf for i in range(NumberOfFrame)]

        print(Frame, '\t # Empty Frame')

        j = 0

        x=0

        PageFault = 0

        for i in range(len(PageReference)):

            if PageReference[i] not in Frame:

                RestList = PageReference[i + 1:len(PageReference)]

                Position = -1

                index = -1
            if np.inf in Frame:

                index = Frame.index(np.inf)

            else:

                for k in range(NumberOfFrame):

                    if Frame[k] in RestList:

                        P = RestList.index(Frame[k])

                        if P > Position:

                            Position = P

                            index = k

                    else:

                        index = k

                        break;

            if index != -1:

                Frame[index] = PageReference[i]

            else:

                Frame[j] = PageReference[i]

                j = (j + 1) % NumberOfFrame

            PageFault += 1

            print(PageReference[i], Frame, '\t occur Page Fault (Miss)')

            mylabel3 = Label(topw, text=Frame, bg="light cyan", font="comicsansms 15
bold", borderwidth=3,relief=RAISED)

            mylabel3.place(x=220, y=140+(40*x))

        else:

            print(PageReference[i], Frame, '\t No Page Fault (Hit)')

            mylabel3 = Label(topw, text=Frame, bg="orange red", font="comicsansms 15
bold", borderwidth=3,relief=SUNKEN)

            mylabel3.place(x=220, y=140+(40*x))

        x+=1

    print('Number of Page Fault: ', PageFault)
```

RESULT:



```
Optimal Algorithm(OPR)

Number of Frame is: 3

Page Reference List is: 3 4 5 3 5 6 7

3 Inf Inf
3 4 Inf
3 4 5
3 4 5
3 4 5
6 4 5
7 4 5

Page Fault:5
```

LIFO Page Replacement Algorithm

LIFO stands for “Last in First out“, and it performs all activities like LIFO principle. In this algorithm, newest page is replaced which is arrived at last in to primary memory, and it uses the stack for monitoring all pages.

LIFO (Last In First Out)

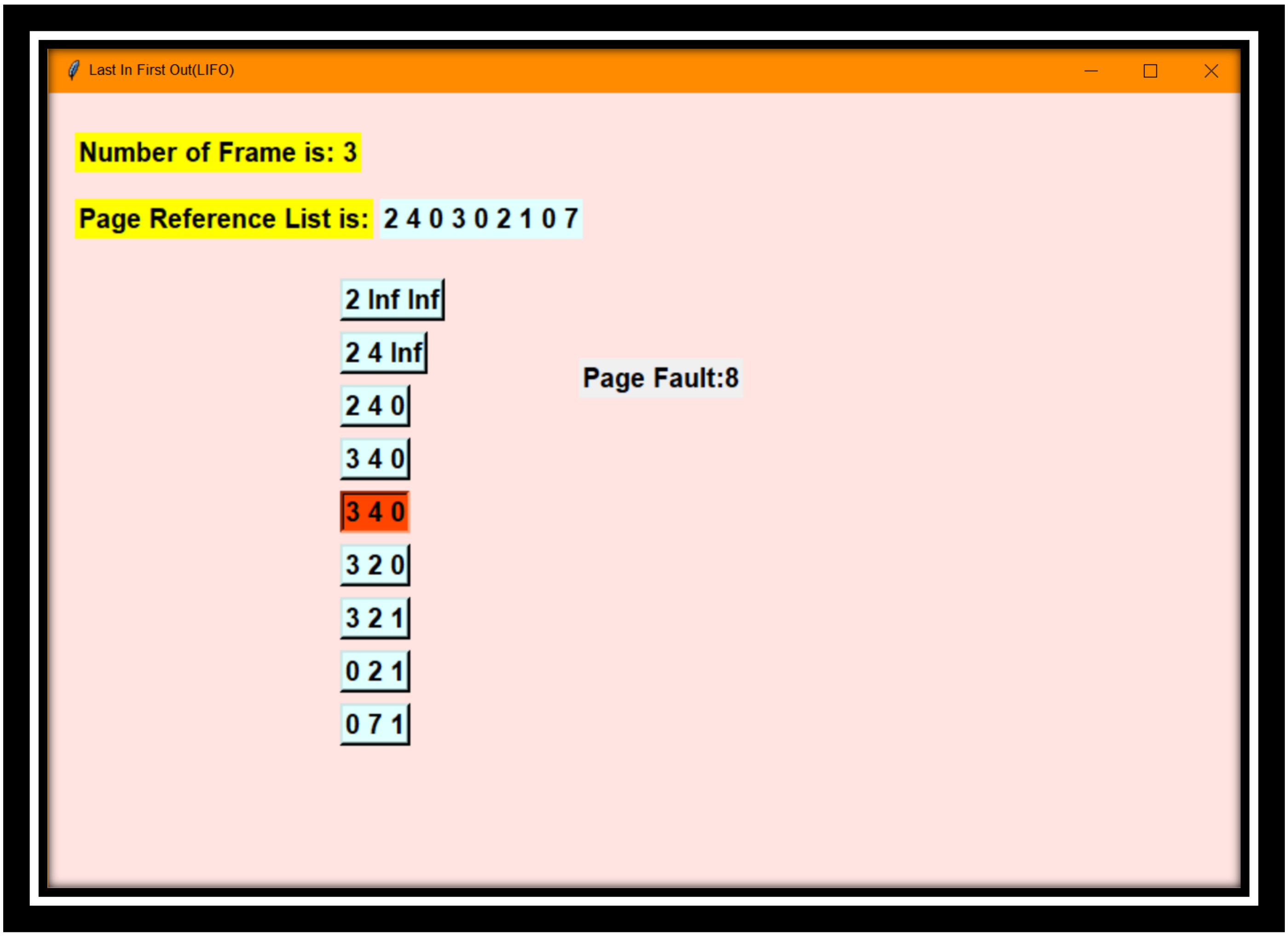
Method: Select the frame that has been in Main Memory the shortest and remove it to make room.

Implementation: maintain a stack of all pages currently in Main Memory and remove the page that is at the top of the stack the linked list.

CODE:

```
def LIFO():
    Frame = [np.inf for i in range(NumberOfFrame)]
    print(Frame, '\t # Empty Frame')
    j = 0
    x = 0
    PageFault = 0
    for i in range(len(PageReference)):
        PreIndex = j

        if PageReference[i] not in Frame:
            PageFault += 1
            Frame[j] = PageReference[i]
            j = (j + 1) % NumberOfFrame
        if PreIndex == j:
            print(Frame, '\t No Page Fault (Hit)')
        mylabel3 = Label(topw, text=Frame, bg="orange red", font="comicsansms 15 bold",borderwidth=3,relief=SUNKEN)

        mylabel3.place(x=220, y=140+(40*x))

    else:

        print(Frame, '\t occur Page Fault (Miss)')

        mylabel3 = Label(topw, text=Frame, bg="light cyan", font="comicsansms 15 bold",borderwidth=3,relief=RAISED)

        mylabel3.place(x=220, y=140+(40*x))

    x+=1

print('Number of Page Fault: ', PageFault)
```

RESULT:



Last In First Out(LIFO)

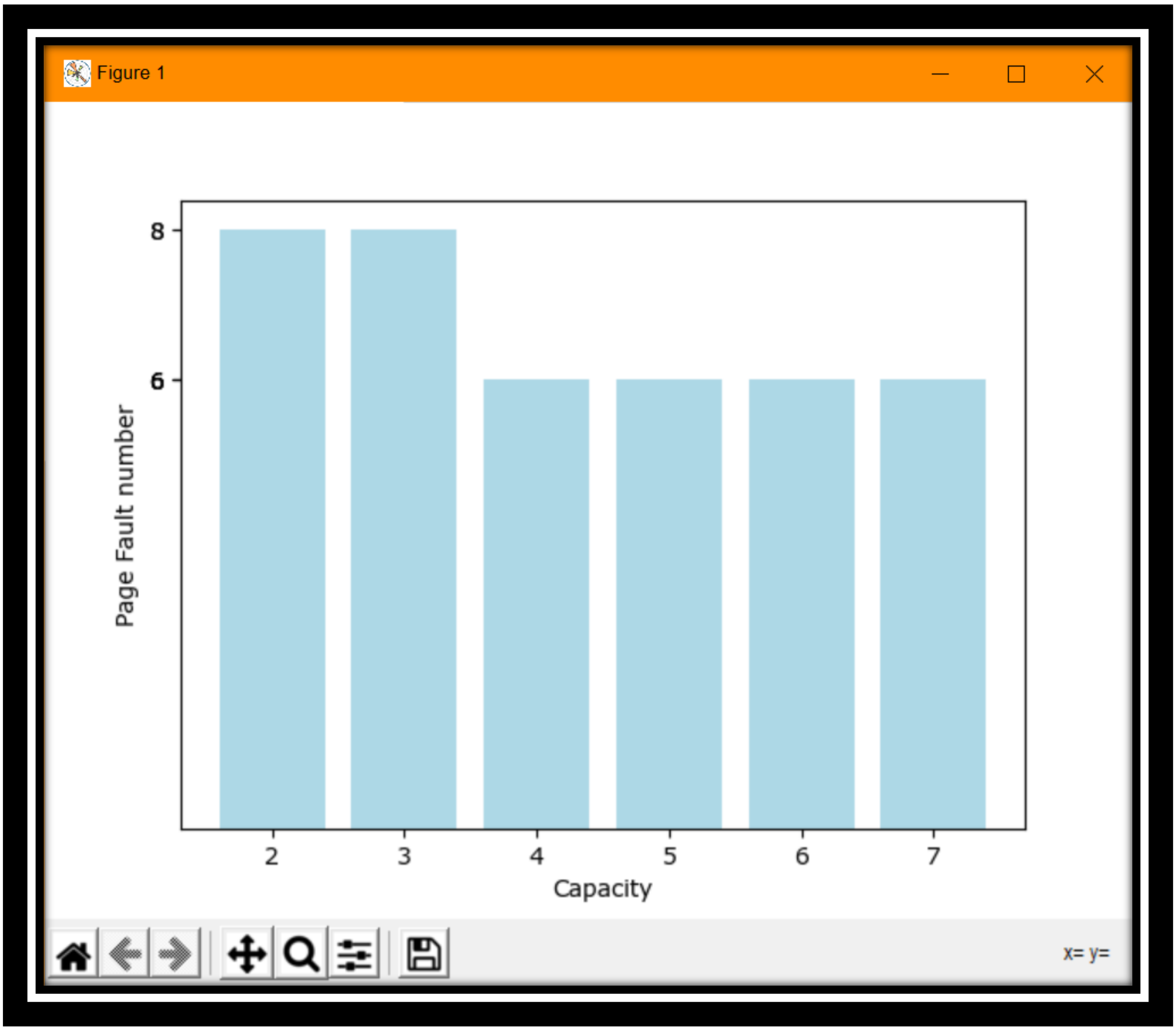
Number of Frame is: 3

Page Reference List is: 2 4 0 3 0 2 1 0 7

```
2 Inf Inf
2 4 Inf
2 4 0
3 4 0
3 4 0
3 2 0
3 2 1
0 2 1
0 7 1
```

Page Fault:8

Graph:



Random Page Replacement Algorithm

In this algorithm, randomly page can be replaced anytime, but it works like other page replacement policy such as FIFO, LIFO, LRU and Optimal.

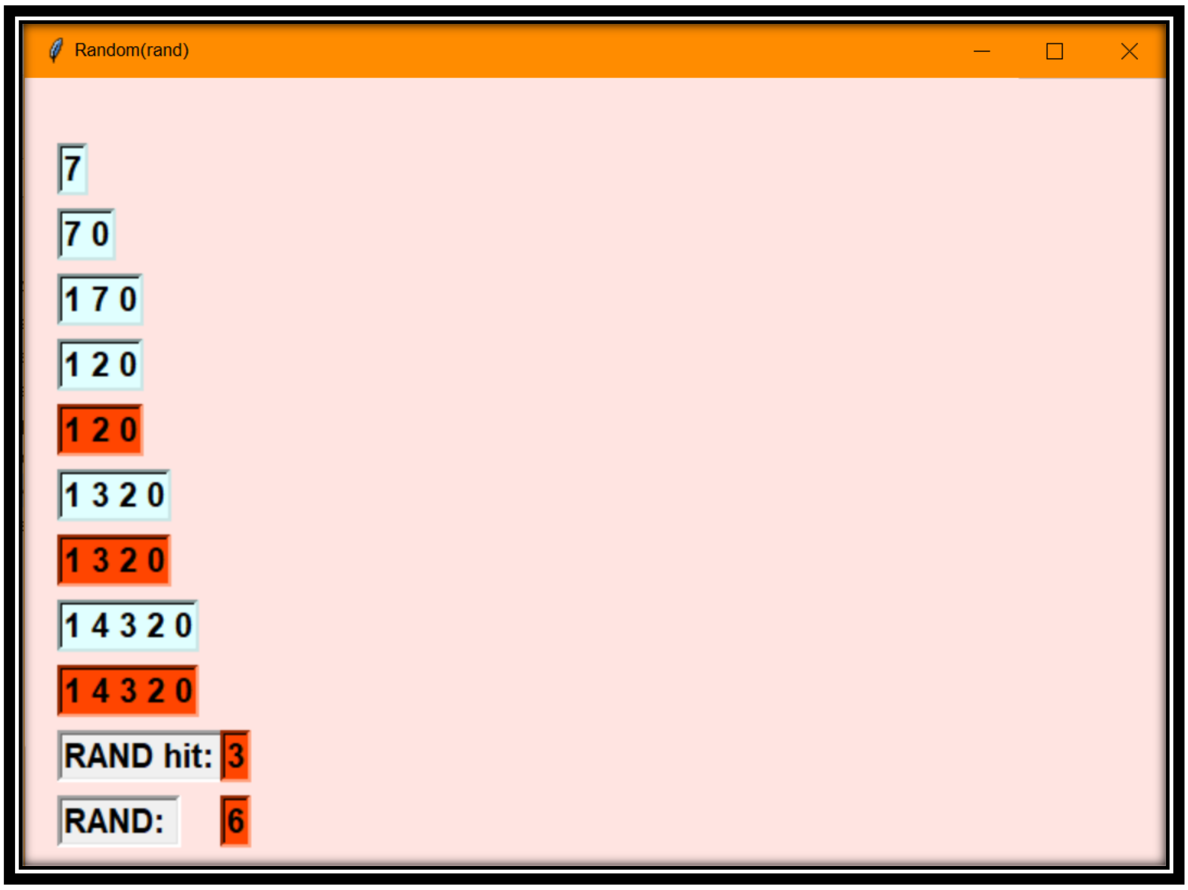
Random Page Replacement

Follow either FIFO, LIFO, LRU and Optimal method and implementation by random selection.

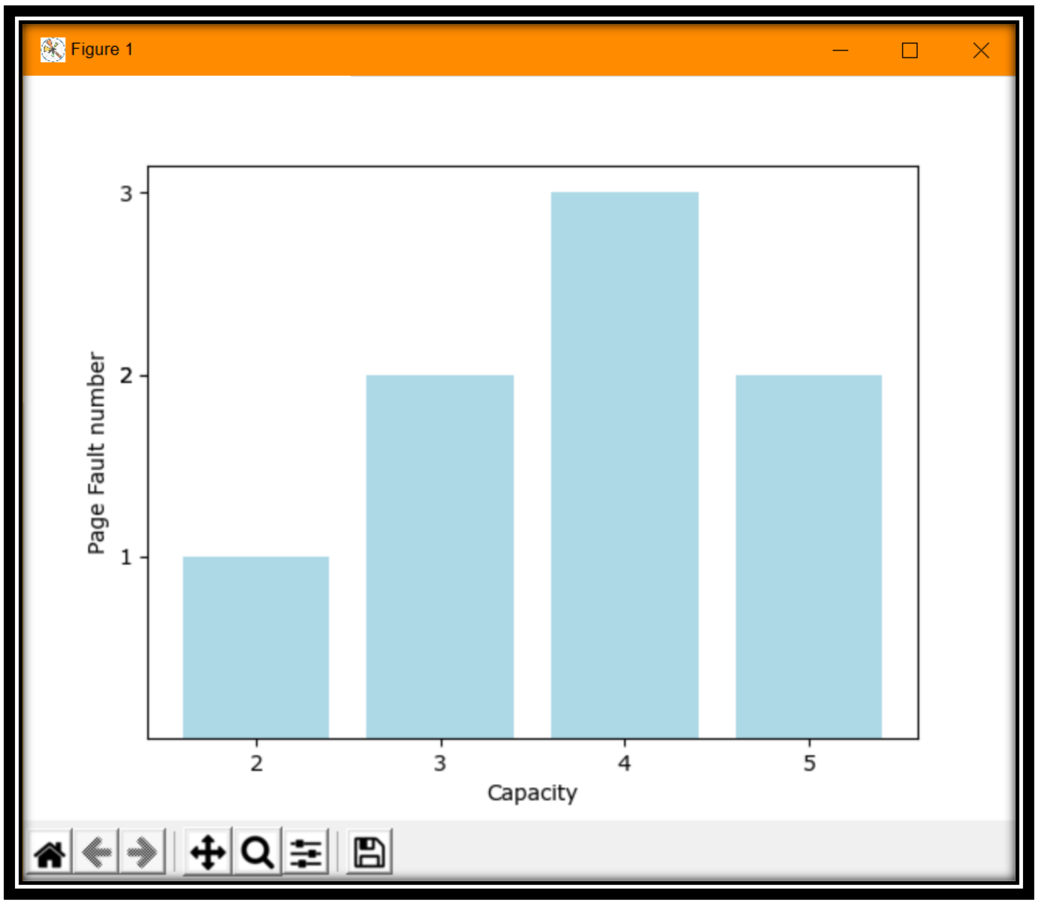
CODE:

```
def rand():
     page_array=list(map(int, e1.get().strip().split(",")))
     frames=int(e.get())
     frame_array = []
     frame_index = 0
     page_faults = 0
     hit=0
     insertions = 0
     x=0
     for index, page in enumerate(page_array):
        if page in frame_array:
            print(frame_array)
            mylabel = Label(topw, text=frame_array, bg="orange red",
     font="comicsansms 15 bold", borderwidth=3,
                             relief=SUNKEN)
            mylabel.place(x=20, y=40 + (40 * x))
            x+=1
            frame_array.insert(toKick, page)
            hit+=1
            continue # should skip to next iteration

        else:
            # Find the victim frame at random
            toKick = randint(0, (frames - 1))
        # Increment the number of page faults
        page_faults = page_faults + 1
        print("page_faults")
        # Put new page into list of frames
        if index < frames:
            frame_array.insert(toKick, page)
            mylabel1 = Label(topw, text=frame_array, bg="light cyan",
font="comicsansms 15 bold", borderwidth=3,relief=SUNKEN)
            mylabel1.place(x=20, y=40 + (40 * x))
            x += 1
            print(frame_array)
        else:
            frame_array[toKick] = page
            mylabel2 = Label(topw, text=frame_array, bg="light cyan",
font="comicsansms 15 bold", borderwidth=3, relief=SUNKEN)
            mylabel2.place(x=20, y=40 + (40 * x))
            x += 1
            print(frame_array)


print("RAND: " + str(page_faults))
print("RAND hit: " + str(hit))
```

RESULT:


Random(rand)

```
7
7 0
1 7 0
1 2 0
1 2 0
1 3 2 0
1 3 2 0
1 4 3 2 0
1 4 3 2 0
RAND hit: 3
RAND:     6
```



Belady’s anomaly

Belady’s anomaly proves that it is possible to have more page faults when increasing the number of page frames while using the First in First Out (FIFO) page replacement algorithm. For example, if we consider reference string 3, 2, 1, 0, 3, 2, 4, 3, 2, 1, 0, 4 and 3 slots, we get 9 total page faults, but if we increase slots to 4, we get 10 page faults.

Reason of Belady’s Anomaly –

The other two commonly used page replacement algorithms are Optimal and LRU, but Belady’s Anamoly can never occur in these algorithms for any reference string as they belong to a class of stack based page replacement algorithms.

Why Stack based algorithms do not suffer Anomaly –

All the stack based algorithms never suffer Belady Anomaly because these type of algorithms assigns a priority to a page (for replacement) that is independent of the number of page frames. Examples of such policies are Optimal, LRU and LFU. Additionally these algorithms also have a good property for simulation, i.e. the miss (or hit) ratio can be computed for any number of page frames with a single pass through the reference string.

Advantages and Disadvantages of generally used Page Replacement algorithms

1. First In First Out (FIFO):

1. Advantages –
    a. It is simple and easy to understand & implement.
2. Disadvantages –
    a. The process effectiveness is low.
    b. When we increase the number of frames while using FIFO, we are giving more memory to processes. So, page fault should decrease, but here the page faults are increasing. This problem is called as Belady’s Anomaly.
    c. Every frame needs to be taken account off.

2. Least Recently Used (LRU):

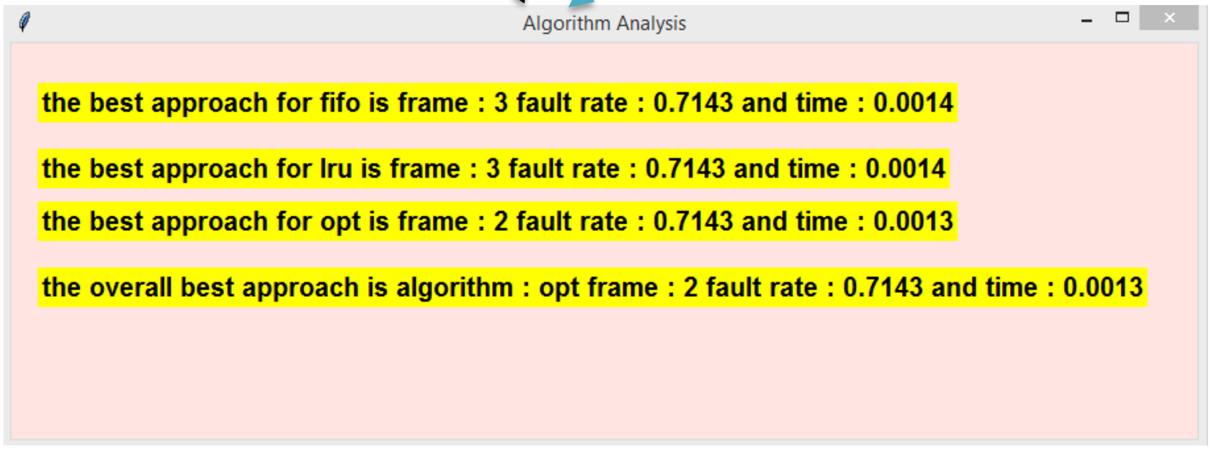
1. Advantages –
   a. It is open for full analysis.
   b. In this, we replace the page which is least recently used, thus free from Belady's Anomaly.
   c. Easy to choose page which has faulted and hasn't been used for a long time.
2. Disadvantages –
   a. It requires additional Data Structure to be implemented.
   b. Hardware assistance is high.

3. Optimal Page Replacement (OPR):

1. Advantages –
   a. Complexity is less and easy to implement.
   b. Assistance needed is low i.e Data Structure used are easy and light.
2. Disadvantages–
   a. OPR is perfect, but not possible in practice as the operating system cannot know future requests.
   b. Error handling is tough.

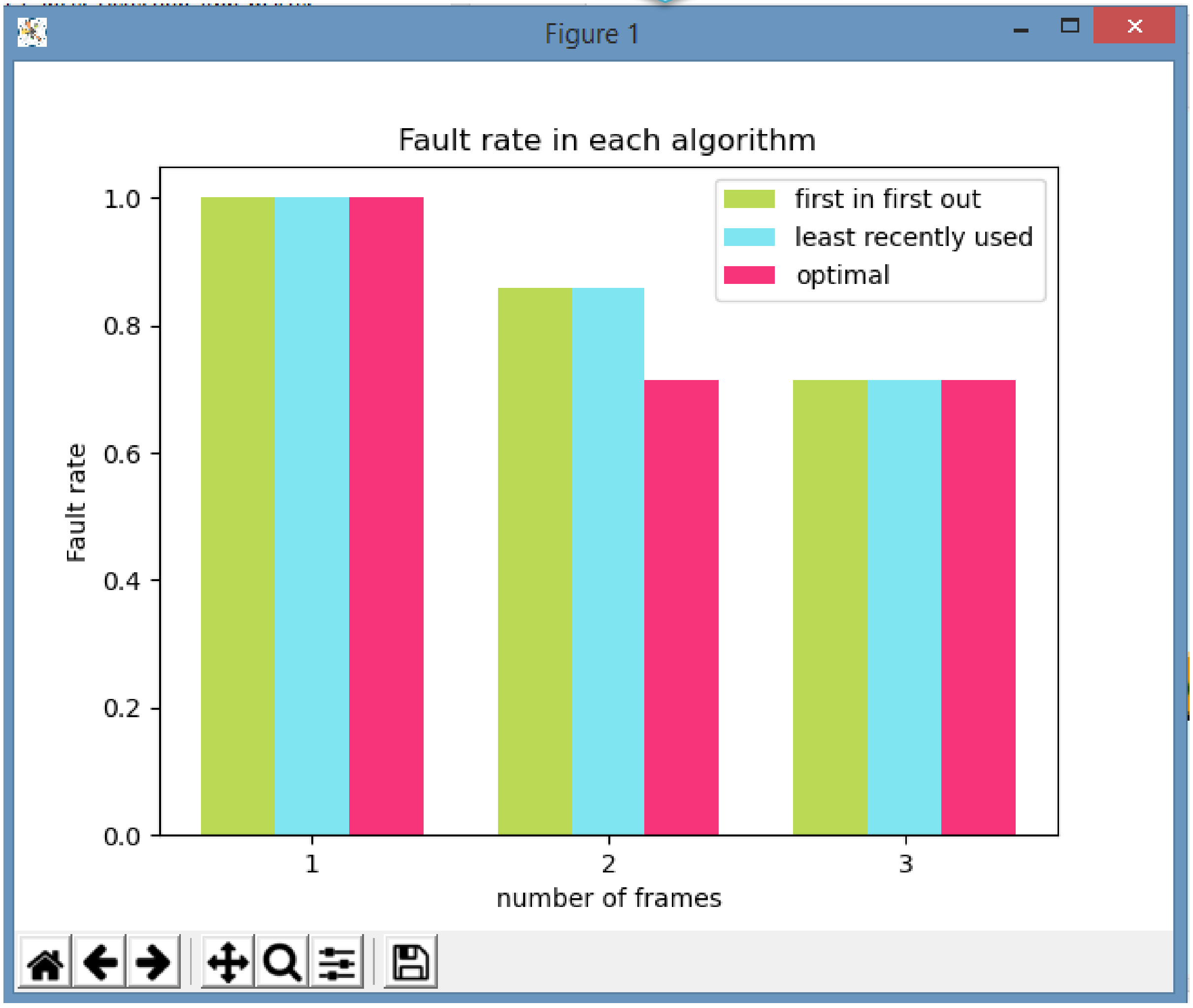
COMPARATIVE ANALYSIS OF GENERALLY USED PAGE REPLACEMENT ALGORITHM

For best algorithm for page fault



```
the best approach for fifo is frame : 3 fault rate : 0.7143 and time : 0.0014
the best approach for lru is frame : 3 fault rate : 0.7143 and time : 0.0014
the best approach for opt is frame : 2 fault rate : 0.7143 and time : 0.0013
the overall best approach is algorithm : opt frame : 2 fault rate : 0.7143 and time : 0.0013
```

Fault rate in algorithm



Frame vs
Page Fault

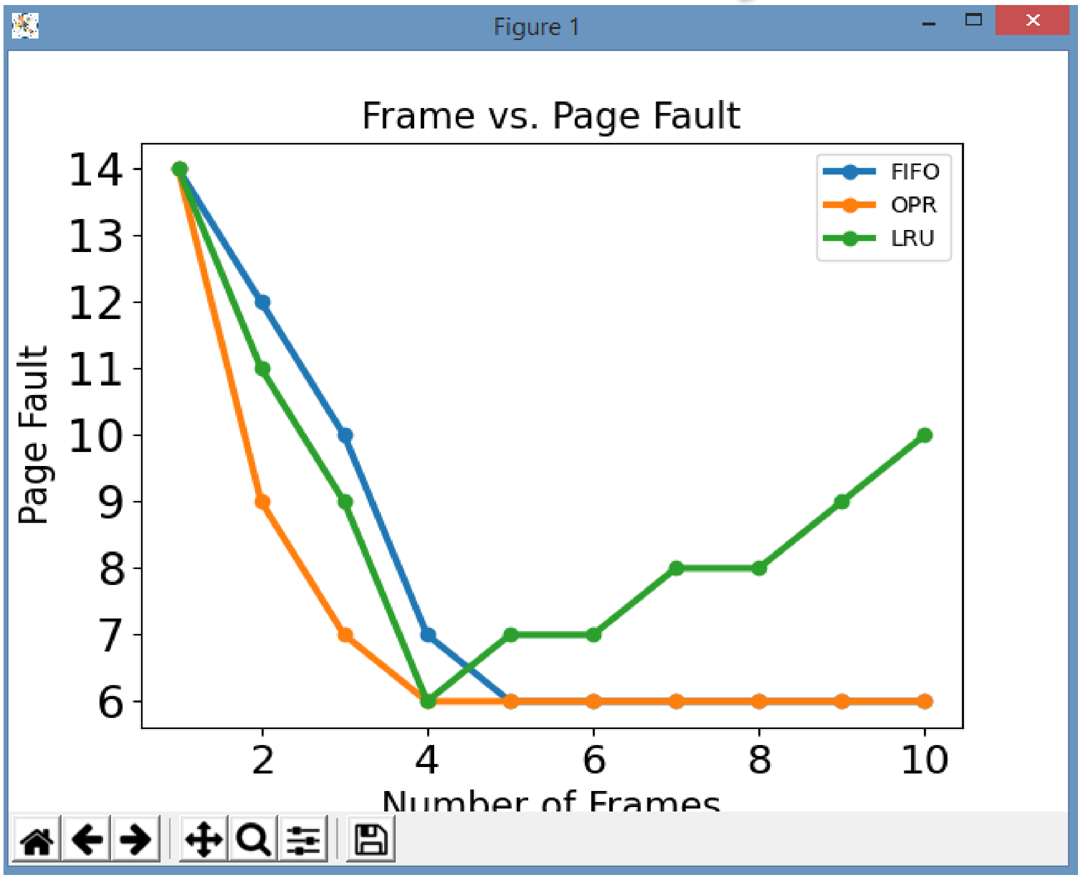


Figure 1
Frame vs. Page Fault
Page Fault
Number of Frames
FIFO
OPR
LRU
14
13
12
11
10
9
8
7
6
2
4
6
8
10

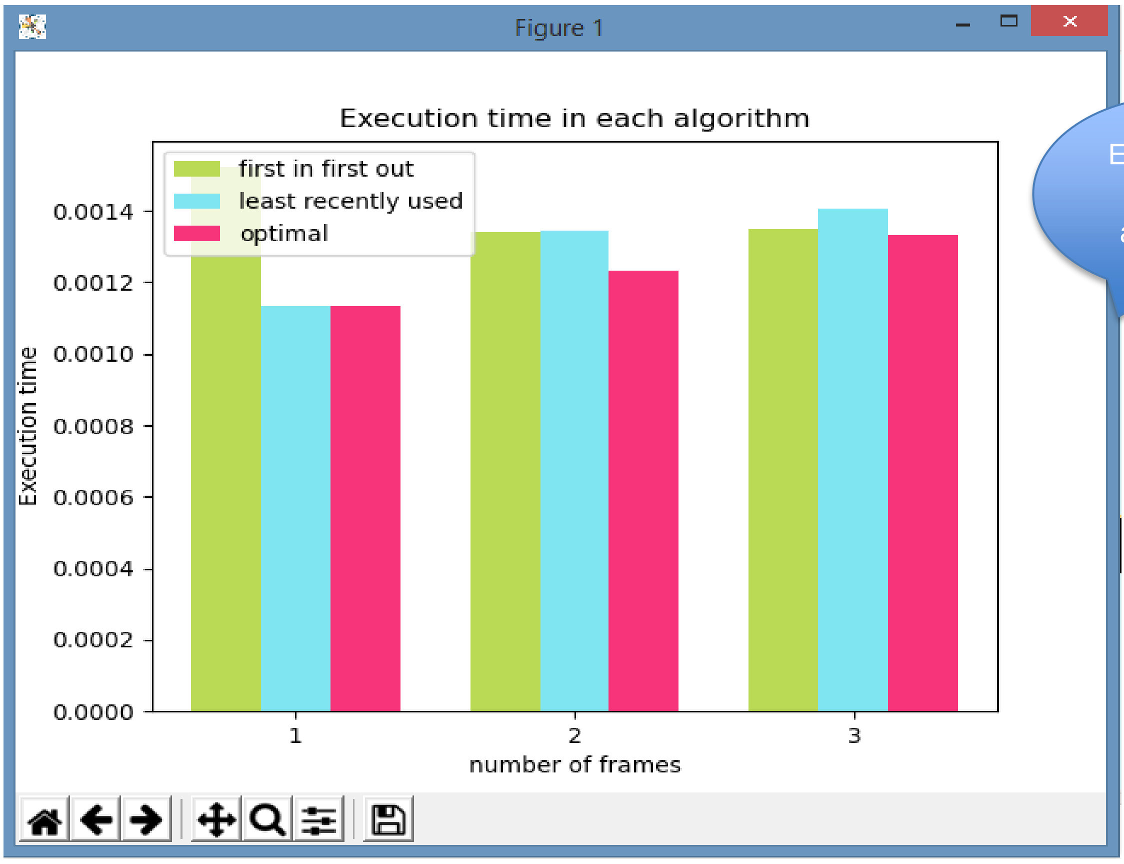


Figure 1
Execution time in each algorithm
first in first out
least recently used
optimal
Execution time
number of frames
0.0014
0.0012
0.0010
0.0008
0.0006
0.0004
0.0002
0.0000
1
2
3

Excecution
time for
algorithm