AOS HW1

M083040017 蘇品瑜

TEST REFERENCE STRING

1. Random

```
import random
len_rs = 100000 # reference string's string
rs_tc1 = [] # reference string
dw_set = [] # disk write set
for i in range(len_rs):
    rs_tc1.append(random.randint(1,500))
    dw_set.append(random.randint(0,1)) # 0: no change,1: dirty
```

2. Locality

```
import random
len_rs = 100000 # reference string's string
rs_tc1 = [] # reference string
dw_set = [] # disk write set

while len(rs_tc1) < len_rs:

    begin = random.randint(1,500)
    maxApend = 500 - begin  # 還可放幾個的扣打
    if maxApend >= 2:
        add = random.randint(1,maxApend)

        if len(rs_tc1) >= len_rs :
            break
        rs_tc1.append(begin)
        dw_set.append(random.randint(0,1))
        for i in range(add):
            begin = begin + 1
            if len(rs_tc1) >= len_rs:
                break
            rs_tc1.append(begin)
            dw_set.append(random.randint(0,1))
    elif maxApend==2:
        if len(rs_tc1) >= len_rs:
            break
        rs_tc1.append(499)
        dw_set.append(random.randint(0,1))
```

```
        if len(rs_tc1) >= len_rs:
            break
        rs_tc1.append(500)
        dw_set.append(random.randint(0,1))
    else:
        rs_tc1.append(500)
        dw_set.append(random.randint(0,1))
```

3. Custom

Why?

搭配 Custom Algorithm 特性，還有和 FIFO 特性唱反調(後進先出)。

Custom Algorithm 特性:

先考慮 reference 在 frame 的累積出現次數，但一旦被替換，則重新計算。

如果一樣，再考慮 誰最近沒被使用。

最後 進行 replacement。

```
import random

rs_tc1 = [] # reference string
len_rs = 100000 # reference string's string

dw_set = [] # disk write set
for i in range(len_rs):
    dw_set.append(random.randint(0,1))

sum = 0 # 第幾個 reference
start = 50 # 一次 (塞(50-1) + 50++)

while sum < len_rs:
    for i in range(1,50):
        rs_tc1.append(i)
        sum = sum + 1
    rs_tc1.append(start)
    start = start + 1
    sum = sum + 1
```

ALGORITHM X REFERENCE STRING = 36 FIGURES & BEHAVIORS

1. FIFO

先進來的 page，在發生 replacement 時，先被替換出去。

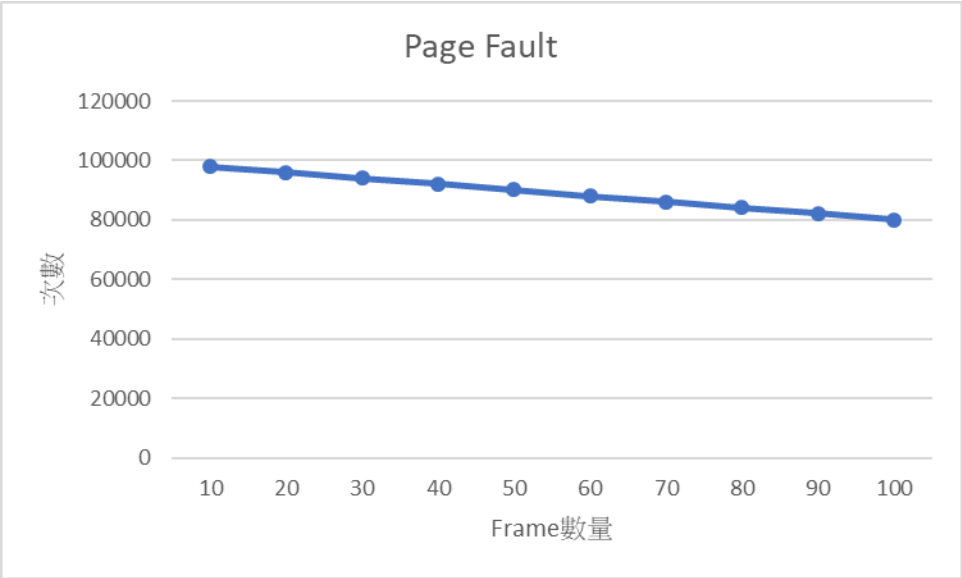
If page not in memory , page fault ++.

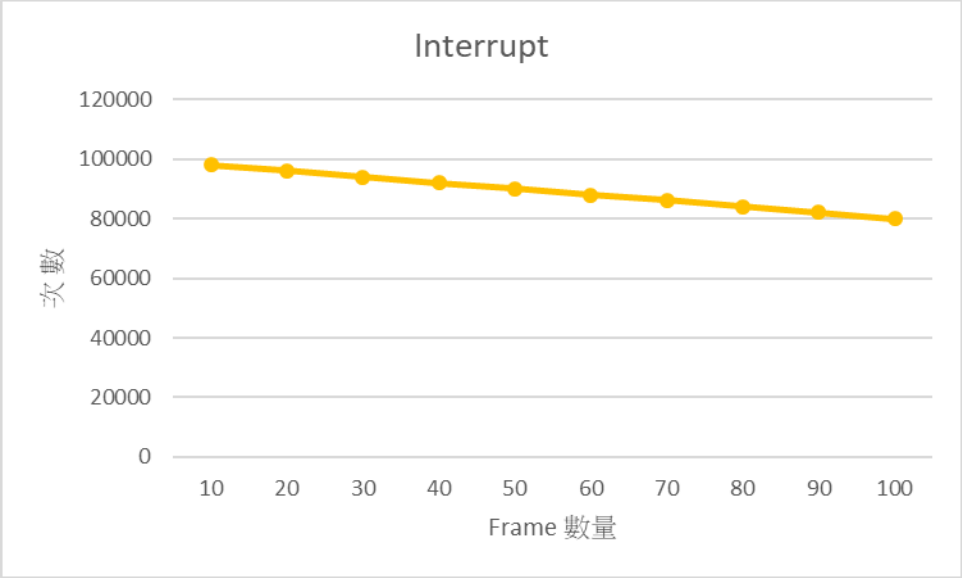
If replacement occurs(swap out), interrupt++.

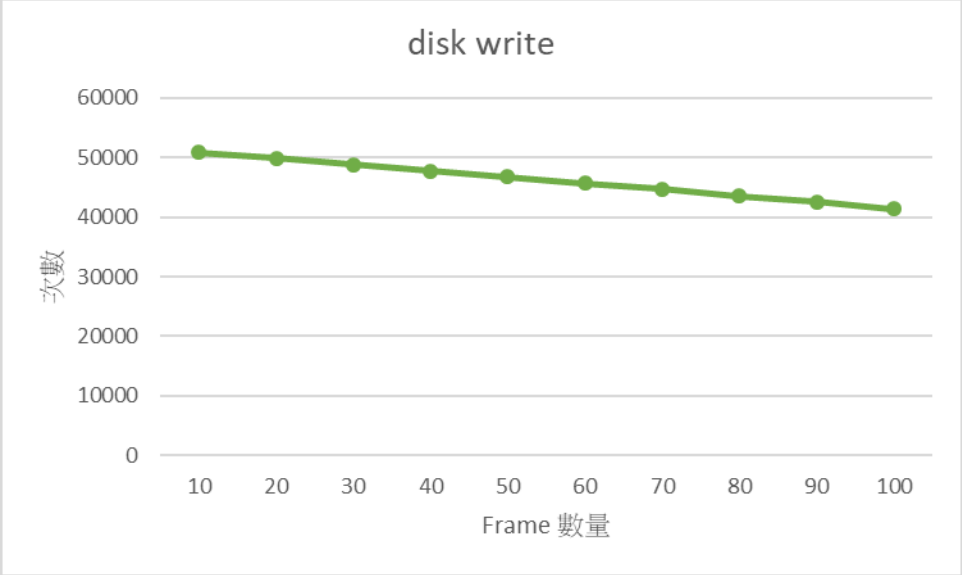
If victim page's dirty bit is 1, disk write++.

```
import queue
for i in range(10):
    frame_number = input("Number of frame: ")
    myqueue = queue.Queue(int(frame_number)) # 設定 frame 數量
    page_fault_sum = 0 # 計算 page fault 數量
    interrupt = 0      # 計算 interrupt 數量
    disk_write = 0     # 計算 disk write 數量


    cnt = 0 # reference string 到第幾個
    # 先塞滿 memory
    while myqueue.full()!=True :
        myqueue.put(random_rs[cnt]) # 將 reference 放入 memory
        cnt = cnt + 1
        page_fault_sum = page_fault_sum + 1

    print('page_fault_sum:',page_fault_sum) # print 目前 page fault 數量



    # 塞滿了，要塞下一個 先看 page 是否在 queue 裡
    # 有 則 hit , 沒有 則 miss

    while cnt != len(random_rs) :
        if random_rs[cnt] in myqueue.queue:
            pass # 不做任何事
        else:
            interrupt = interrupt + 1 # swap out occurs
            page_fault_sum = page_fault_sum + 1
            if dw_set[random_rs.index(myqueue.get())] == 1 : # modify
                disk_write = disk_write + 1

            myqueue.put(random_rs[cnt]) # 放入新進的 reference
        cnt = cnt + 1

    print('page_fault_sum: ',page_fault_sum)
    print('interrupt: ',interrupt)
    print('disk write: ',disk_write)
```

>fifo_Random_rs.py

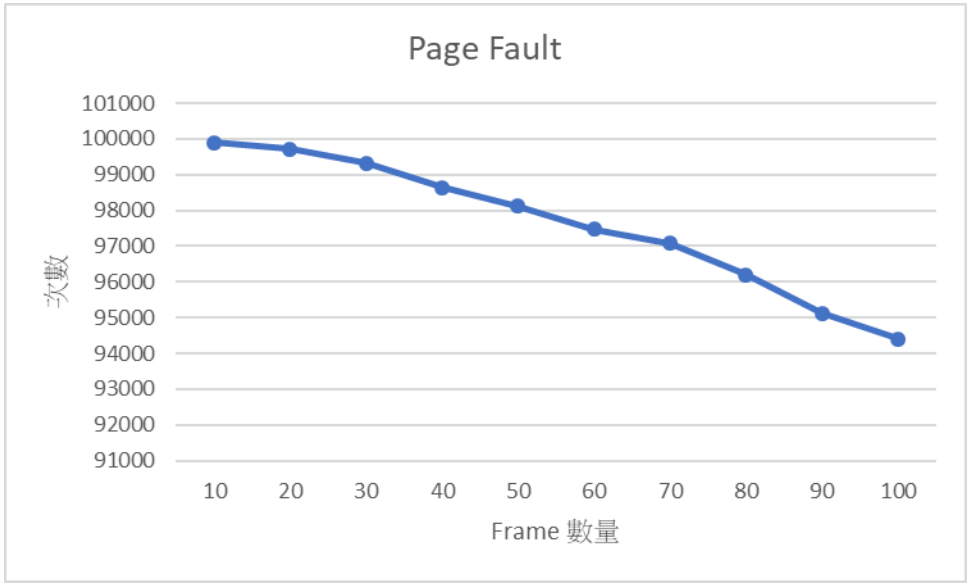


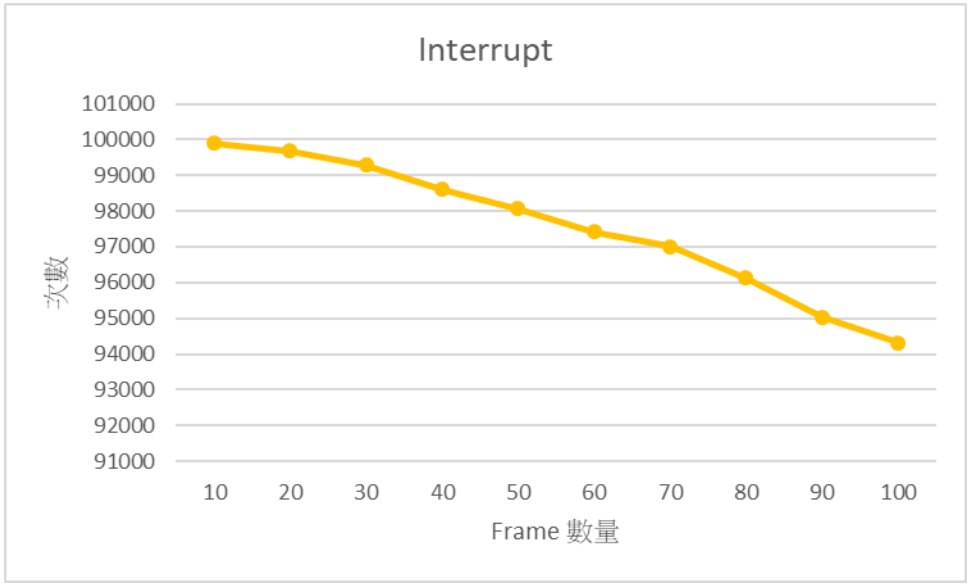


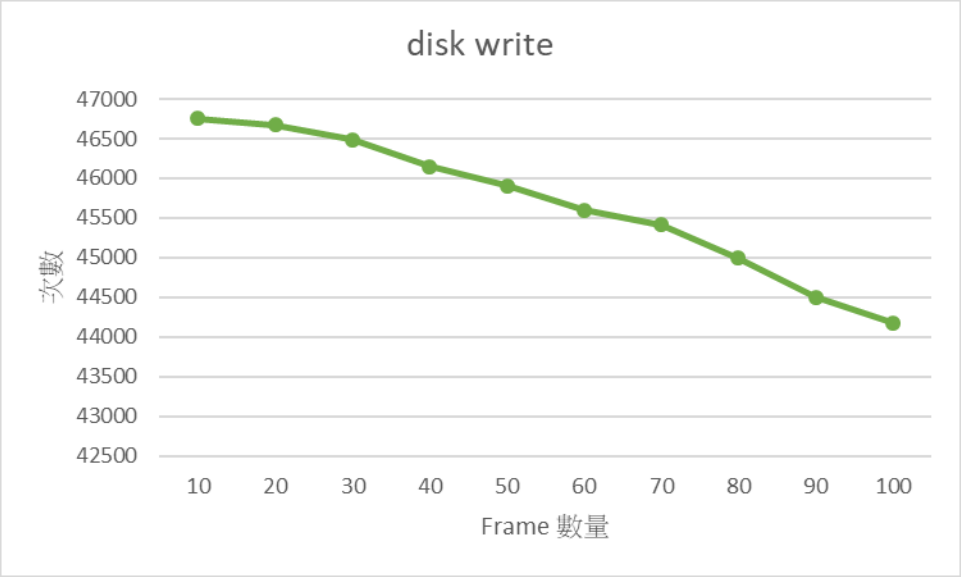


	Page Fault	Interrupt	disk write
10	98001	97991	50865
20	96084	96064	49875
30	94031	94001	48820
40	92090	92050	47777
50	90110	90060	46746
60	88016	87956	45667
70	86155	86085	44687
80	84113	84033	43578
90	82144	82054	42551
100	80022	79922	41425

>fifo_Locality_rs.py



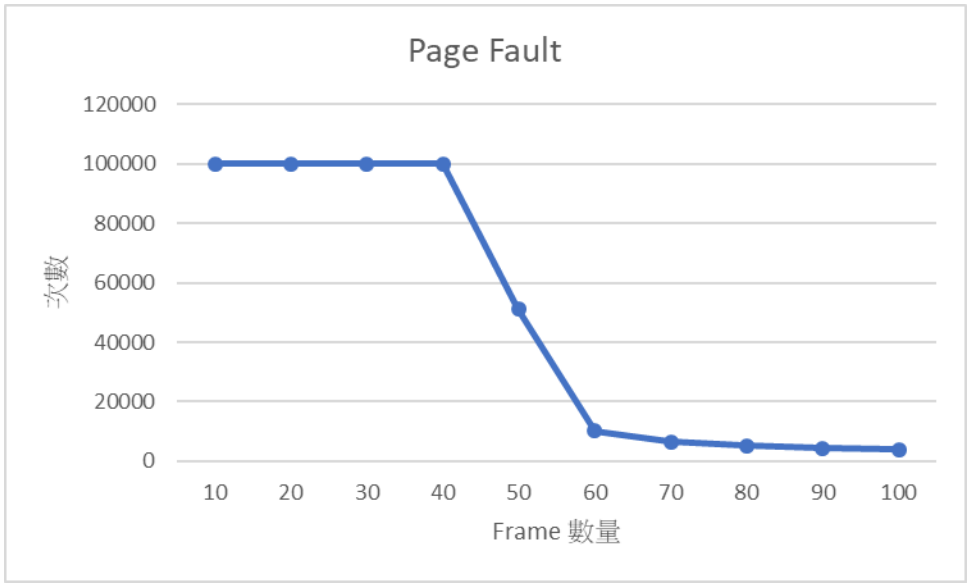


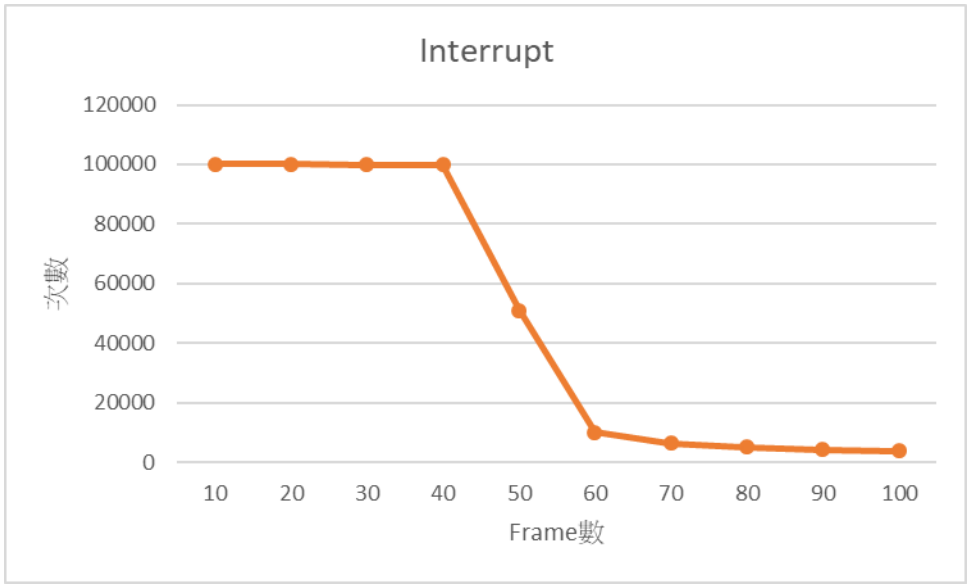


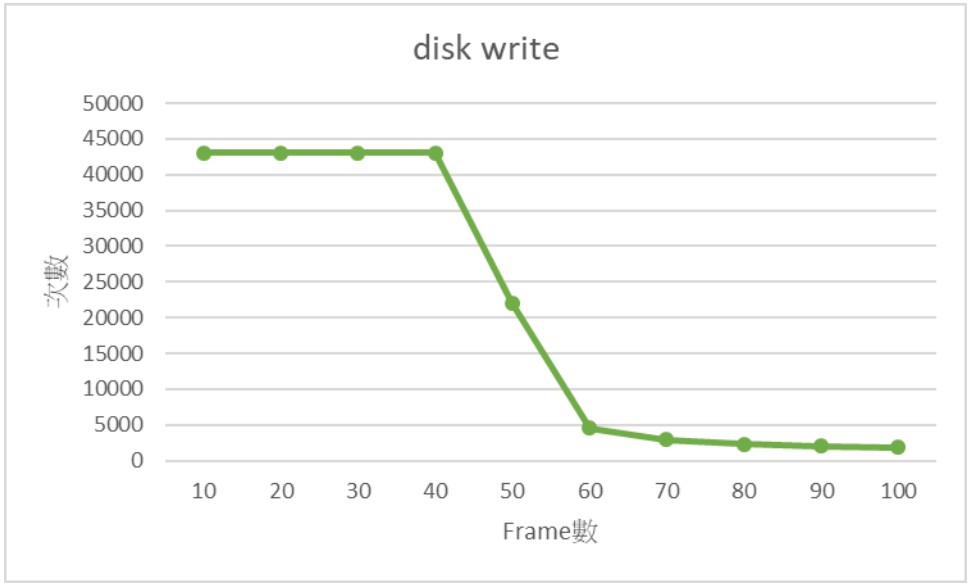
	Page Fault	Interrupt	disk write
10	99903	99893	46755
20	99710	99690	46674
30	99321	99291	46485
40	98642	98602	46146
50	98123	98073	45905
60	97481	97421	45600
70	97083	97013	45416
80	96204	96124	44996
90	95128	95038	44500
100	94406	94306	44180

	Page Fault	Interrupt	disk write
10	99903	99893	46755
20	99710	99690	46674
30	99321	99291	46485
40	98642	98602	46146
50	98123	98073	45905
60	97481	97421	45600
70	97083	97013	45416
80	96204	96124	44996
90	95128	95038	44500
100	94406	94306	44180

>fifo_custom.py


Page Fault
120000
100000
80000
60000
40000
20000
0
次數
10
20
30
40
50
60
70
80
90
100
Frame 數量


Interrupt
120000
100000
80000
60000
40000
20000
0
次數
10
20
30
40
50
60
70
80
90
100
Frame數


disk write
50000
45000
40000
35000
30000
25000
20000
15000
10000
5000
0
次數
10
20
30
40
50
60
70
80
90
100
Frame數

	Page Fault	Interrupt	disk write
10	100000	99990	43023
20	100000	99980	43018
30	100000	99970	43015
40	100000	99960	43010
50	51000	50950	22005
60	10193	10133	4513
70	6479	6409	2916
80	5117	5037	2326
90	4392	4302	2013
100	3960	3860	1824

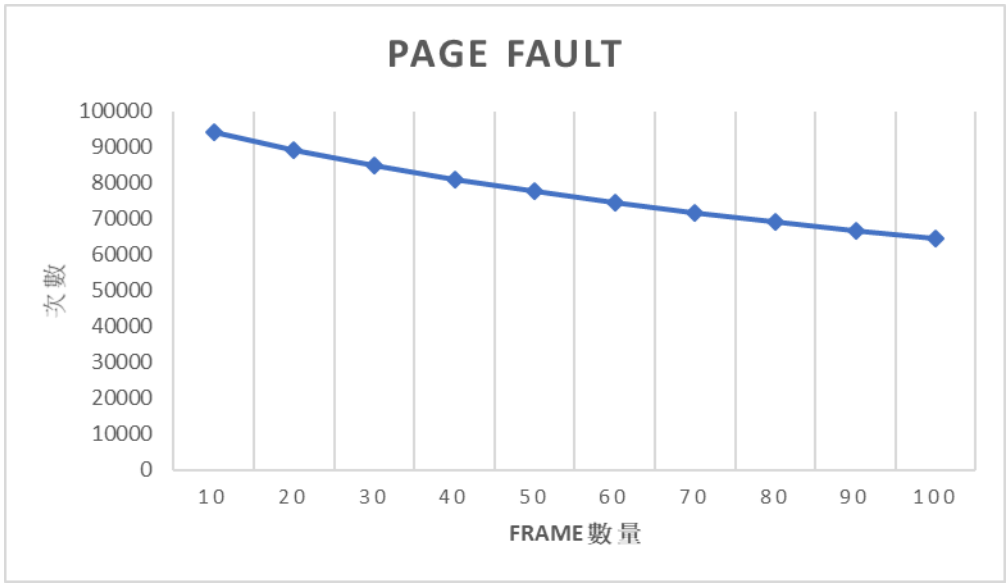
2. Optimal

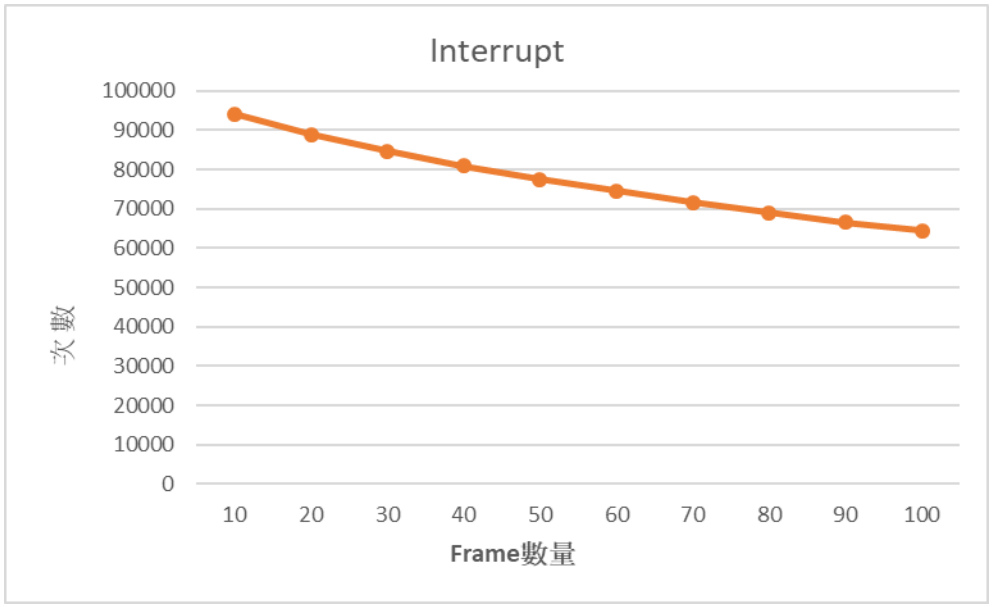
假設可以預知未來的 reference，當發生替換時，則選擇未來不會用或是較晚才會用到的 page，作為 victim page。

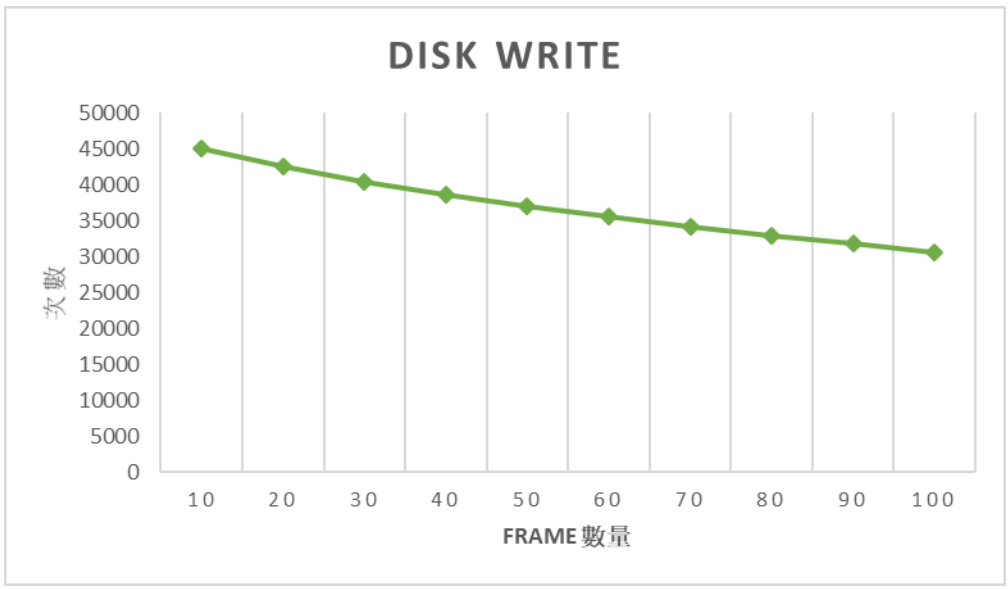
```
for i in range(10):
    fn_tc1 = input("輸入 frame 數: ")
    fn_tc1 = int(fn_tc1)
    hits_tc1 = 0
    misses_tc1 = 0

    frame = []
    frame_record = [] # 紀錄 未來出現情況
    # rs 每往前一步 距離-- , 只更新新進來的 page 距離(不需重新計算)
    cnt   = 0 # 第幾個 frame
    index = 0 # rs_tc1's index
    page_fault = 0
    interrupt  = 0
    dw = 0

    # 塞滿 memory
    while cnt <  fn_tc1 :
        frame.append(rs_tc1[index])
        frame_record.append(int(0)) # 使得 與 frame 數量一致，default 放 0

        cnt = cnt + 1
        page_fault = page_fault + 1
        # 統計 距離下次 reference 出現，如果沒有設 100000+1
        for i in range(len(frame)):
            if frame[i] in rs_tc1[index+1:len(rs_tc1)]:
                frame_record[i] = rs_tc1[index+1:len(rs_tc1)].index(frame[i])
- index
            else:
```

```
                frame_record[i] = (100000+1)
        index = index + 1




    print(page_fault) # 顯示目前 page fault 數

    while index <= (len(rs_tc1) - 1):
        flag = 0
        if rs_tc1[index] in frame:
            pass # do nothing
        else:
            # 挑一個 距離最遠的  # 記得 挑完，全部距離--
            victim_page_index = frame_record.index(max(frame_record))

            if dw_set[rs_tc1.index(frame[victim_page_index])] == 1:
                dw = dw + 1
            # 全部距離 --
            for k in range(fn_tc1):
                frame_record[k] = frame_record[k] - 1
            # 取代
            frame[victim_page_index] = rs_tc1[index]

            # 計算新進來 的 page 的距離
            # 因為 已取代，所以 frame[victim_page_index] 代表 新進 frame
            if frame[victim_page_index] in rs_tc1[index+1:len(rs_tc1)]:
                frame_record[victim_page_index] = rs_tc1[index+1:len(rs_tc1)].
index(frame[victim_page_index]) - index
            else:
                frame_record[victim_page_index] = (100000+1)

            page_fault = page_fault + 1
            interrupt = interrupt + 1
        index = index + 1

    print('Final Page Fault: ',page_fault)
    print('interrupt: ',interrupt)
    print('disk write: ',dw)
```

>optimal_rrs.py

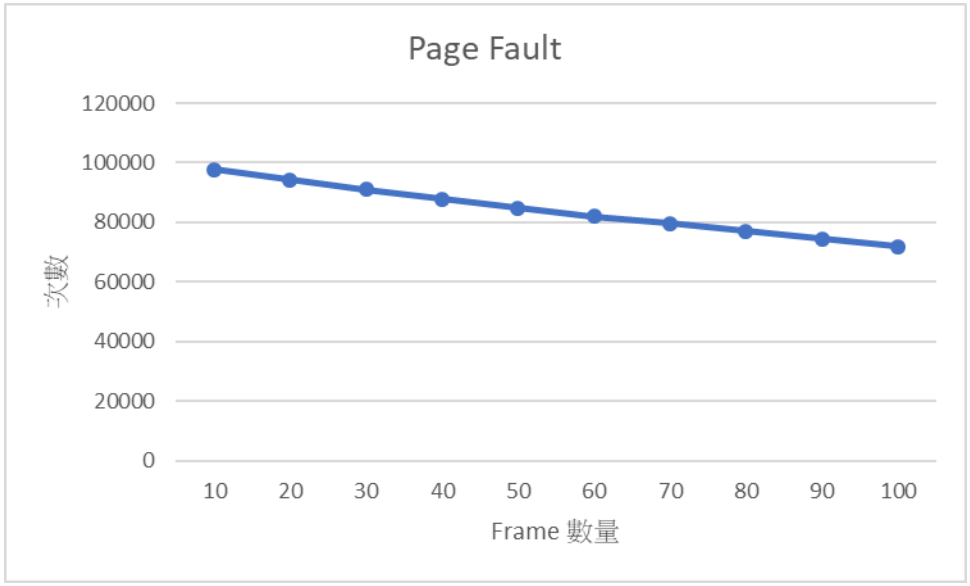

PAGE FAULT
100000
90000
80000
70000
60000
50000
40000
30000
20000
10000
0
次數
10
20
30
40
50
60
70
80
90
100
FRAME數量

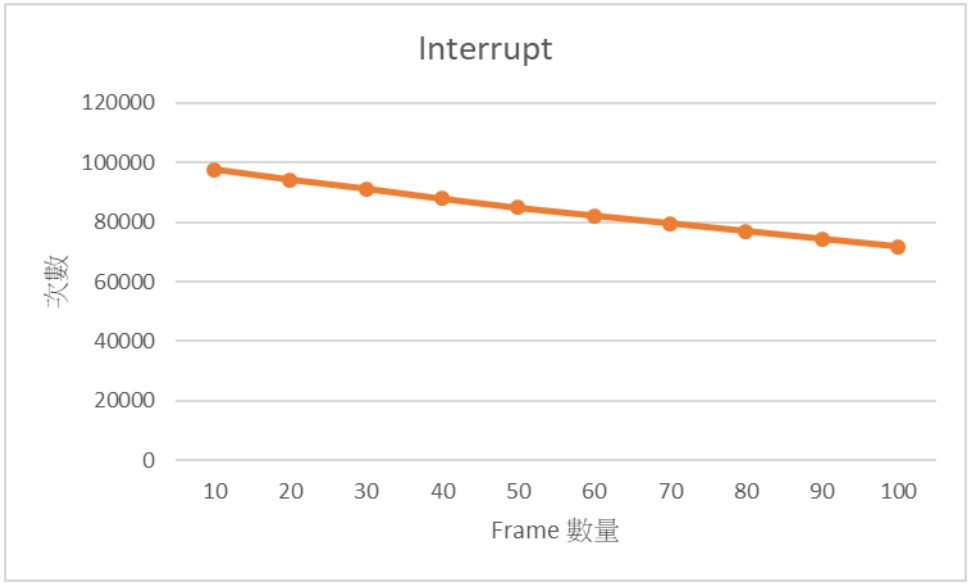

Interrupt
100000
90000
80000
70000
60000
50000
40000
30000
20000
10000
0
次數
10
20
30
40
50
60
70
80
90
100
Frame數量

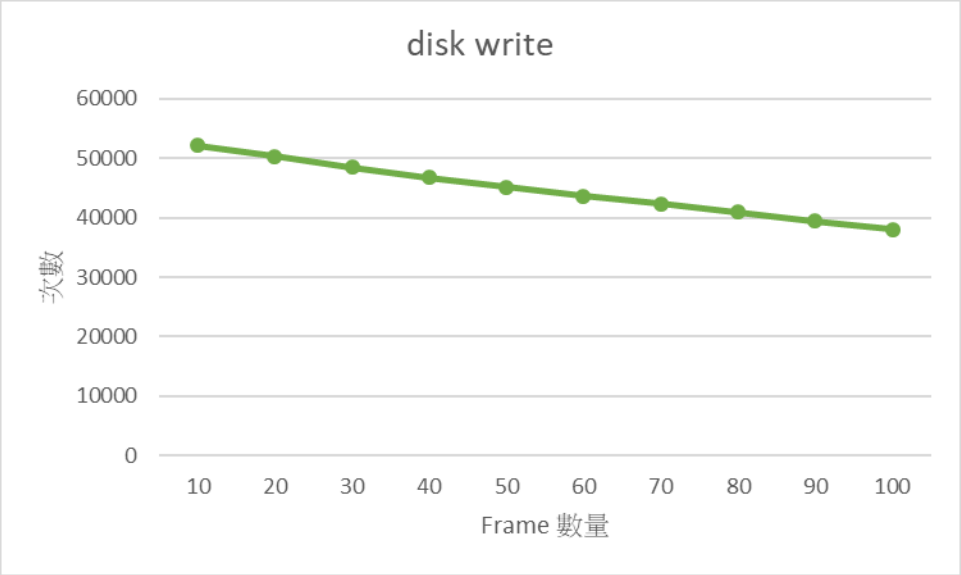

DISK WRITE
50000
45000
40000
35000
30000
25000
20000
15000
10000
5000
0
次數
10
20
30
40
50
60
70
80
90
100
FRAME數量

	Page Fault	Interrupt	disk write
10	94156	94146	45015
20	89017	88997	42511
30	84747	84717	40428
40	80898	80858	38630
50	77551	77501	37018
60	74586	74526	35572
70	71770	71700	34226
80	69162	69082	32945
90	66726	66636	31780
100	64473	64373	30668

>optimal_lrs.py

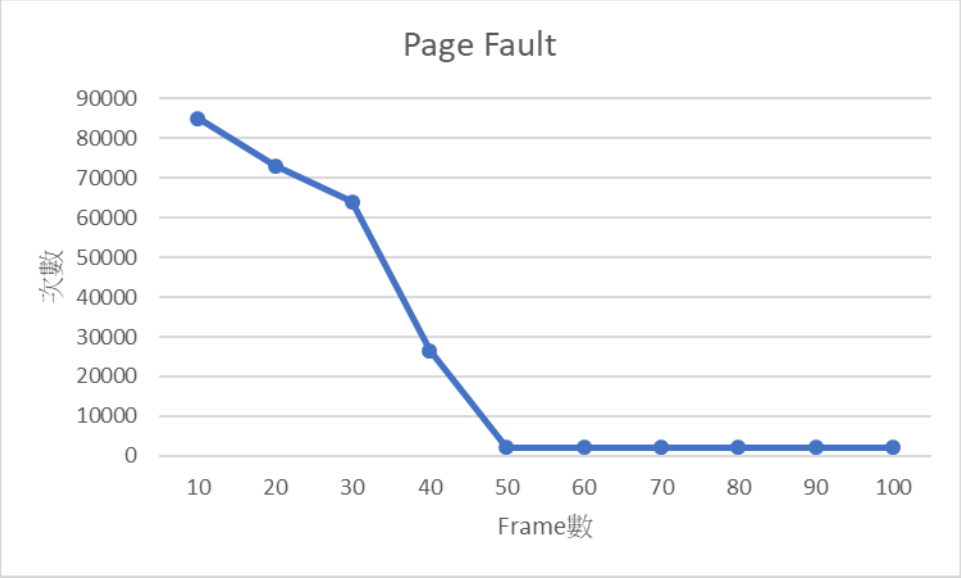


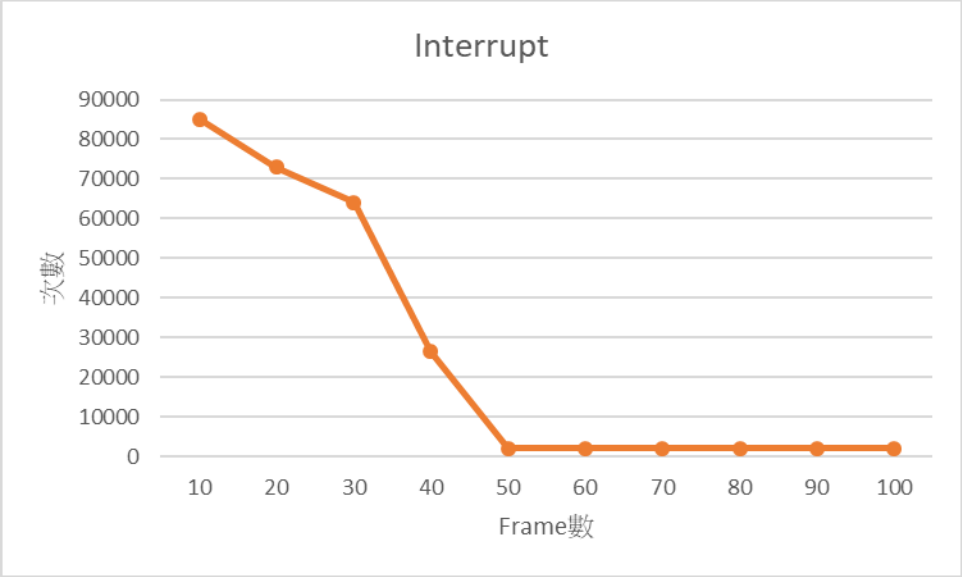


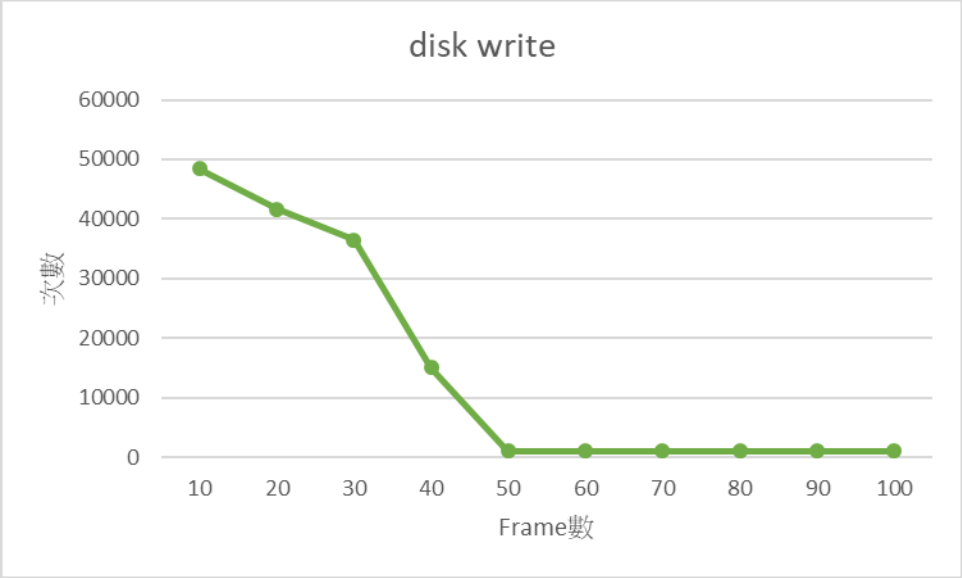


	Page Fault	Interrupt	disk write
10	97630	97620	52143
20	94286	94266	50296
30	91094	91064	48508
40	87894	87854	46734
50	84852	84802	45107
60	82066	82006	43617
70	79612	79542	42296
80	77068	76988	40944
90	74377	74287	39462
100	71862	71762	38071

>optimal_custom.py







	Page Fault	Interrupt	disk write
10	85051	85041	48474
20	73020	73000	41593
30	64027	63997	36444
40	26529	26489	15014
50	2049	1999	1021
60	2059	1999	1021
70	2069	1999	1021
80	2079	1999	1021
90	2089	1999	1021
100	2098	1998	1020

3. ESC (circular)

Second chance 進化版。

會有個 pointer ，輪流指 frame，被指到的且 reference bit 為 0，則替換。

Ex : frame 數為 3 ，指的順序為 0,1,2,0,1,2…以此類推

第一次進去 memory 時，reference bit 設 0，如果連續出現，reference bit 設 1。

如果是 1，得到一次 pass 權(可不被替換掉)。

如果 pointer 指到 ，且 reference bit 為 1，pass，但 reference 被設為 0。

多了一個 modify bit 要記錄。

Random 產生 reference 對應的 dirty bit，If is 1, modify. If is 0, no change.

[程式運作]

從 index 0 開始，找 reference bit 為 0 且 dirty bit 為 0 的情況，優先取代。

有找到則 reference string 換下一位。

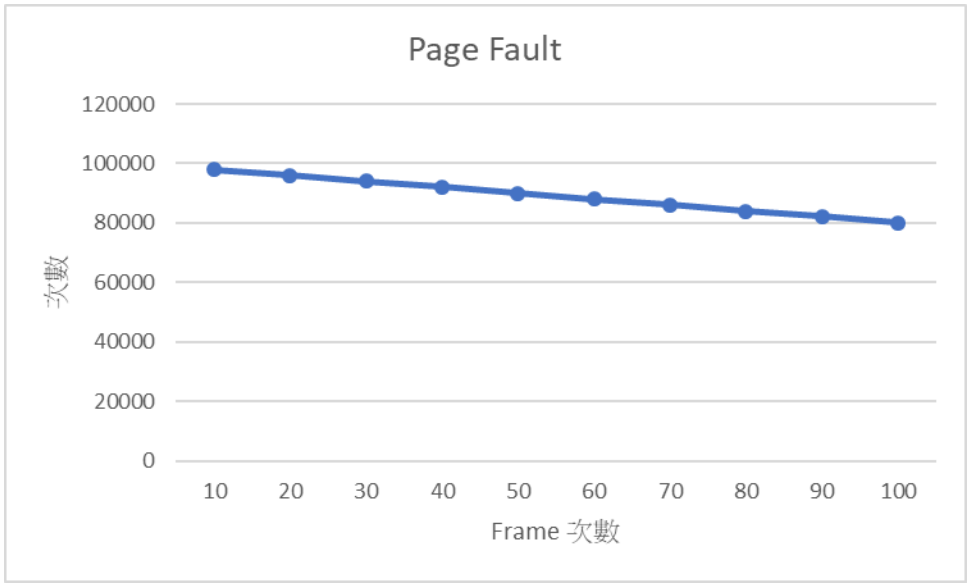
沒有找到 則找 reference bit 為 0 且 dirty bit 為 1 的，做取代，經過但 reference bit 的 page 為 1 則 reset 為 0。

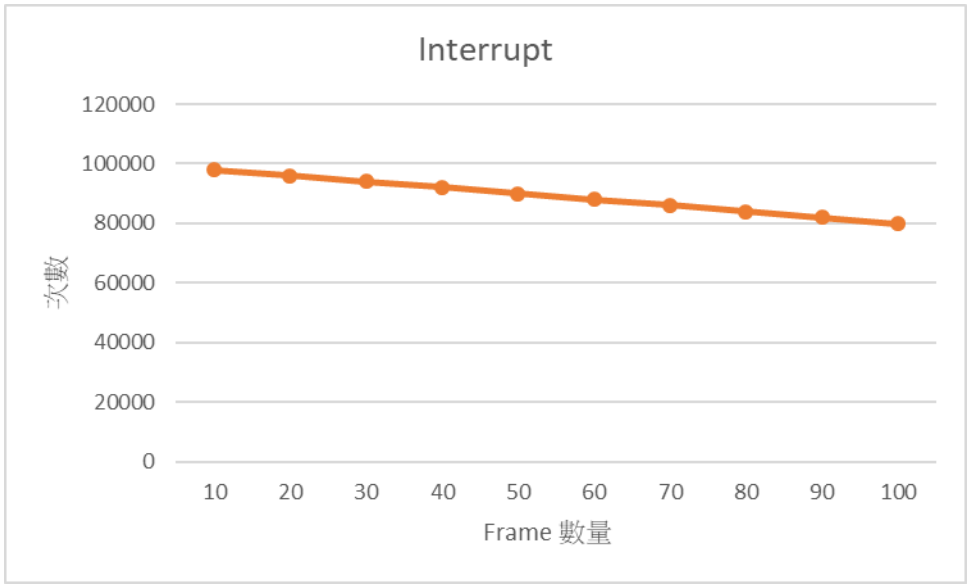
```
for k in range(10):
    frame_number=input("輸入 frame 數量")
    frame_number = int(frame_number)
    index = 0
    page_fault = 0
    interrupt = 0
    dw = 0
    pointer = 0

    # Roud 1 - fill in
    for i in range(frame_number):
        if modify[index] == 1:
            Frame[i] = {"value":rs[index],"r":0,"m":1}
        else:
            Frame[i] = {"value":rs[index],"r":0,"m":0}
        page_fault = page_fault + 1
        index = index + 1

    print("fill in ",page_fault)

    # one cycle back to position 0 : point to 0
    # pointer = 0


    # Round 2 - judge whether page fault
```

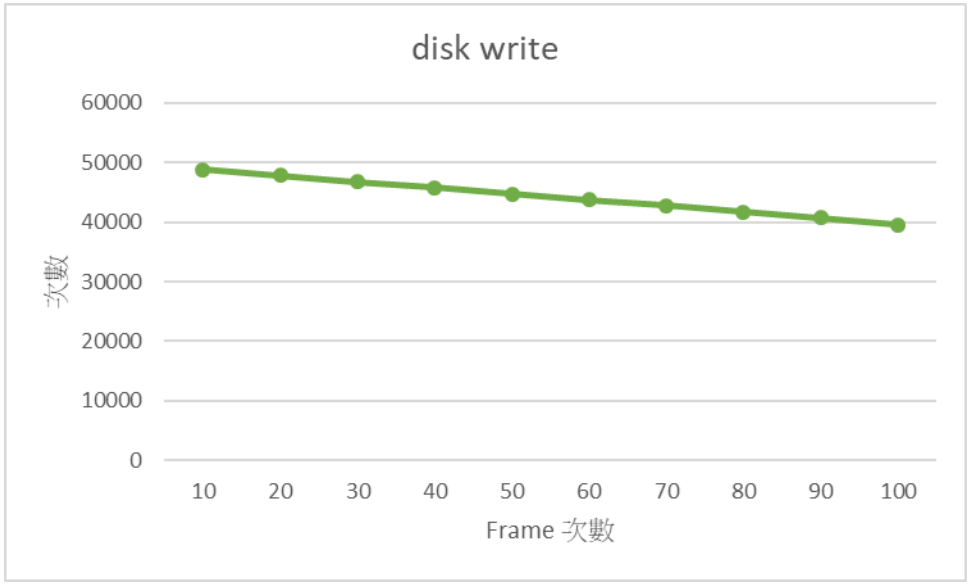
```
while index < len(rs):

    frame_status = [] # 紀錄 該 frame 存放的 value
    status       = [] # reference bit: r, dirty bit: m


    for i in range(frame_number):
        # 印出目前狀況
        #print(Frame[i]["value"]," ",Frame[i]["r"]," ",Frame[i]["m"])
        frame_status.append(Frame[i]["value"])
        status.append([Frame[i]["r"],Frame[i]["m"]])
    # check whether in frame
    exist = 0 # default is not found

    for i in range(frame_number):
        if rs[index] == Frame[i]["value"]:
            Frame[i]["r"] = 1
            exist = 1 # found
            index = index + 1
            break # break search

    # page fault 發生 shift to next page
    # or page survive shift to next page
    if exist == 0 :

        while True: # cycle frame list

            #print('pointer: ',pointer)
            out_of_Loop1 = 0
            # run Loop 1
            while True:

                if Frame[pointer]["r"] == 0 and Frame[pointer]["m"] == 0:
                    Frame[pointer]["value"] = rs[index]
                    Frame[pointer]["m"] = modify[index]
                    Frame[pointer]["r"] = 0
                    page_fault = page_fault + 1
                    interrupt = interrupt + 1
                    index = index + 1
                    out_of_Loop1 = 1
                    pointer = (pointer%4) + 1
                    break # Loop1
                else:
                    pointer = pointer + 1

                if pointer >= (frame_number-1):
                    pointer = (pointer%4) + 1
```

```
                        break # Loop1




                if out_of_Loop1 == 1 :
                    break    # jump out While True == 不用再找下去

                # run Loop 2
                out_of_Loop2 = 0
                if out_of_Loop1 == 0:
                    while True:
                        if Frame[pointer]["r"] == 0 and Frame[pointer]["m"] ==
1:

                            Frame[pointer]["value"] = rs[index]
                            page_fault = page_fault + 1
                            interrupt = interrupt + 1
                            dw = dw + 1
                            Frame[pointer]["m"] = modify[index]
                            Frame[pointer]["r"] = 0
                            out_of_Loop2 = 1
                            index = index + 1

                            pointer = (pointer%4) + 1
                            break # Loop 2
                        else:

                            Frame[pointer]["r"] = 0
                            pointer = pointer + 1

                        if pointer >= (frame_number-1) :
                            pointer = (pointer%4) + 1
                            break

                if out_of_Loop2 == 1 :
                    break   # jump out While True == 不用再找下去



    print("total page fault ",page_fault)
    print("interrupt ",interrupt)
    print("dw ",dw)
```

>esc_random.py

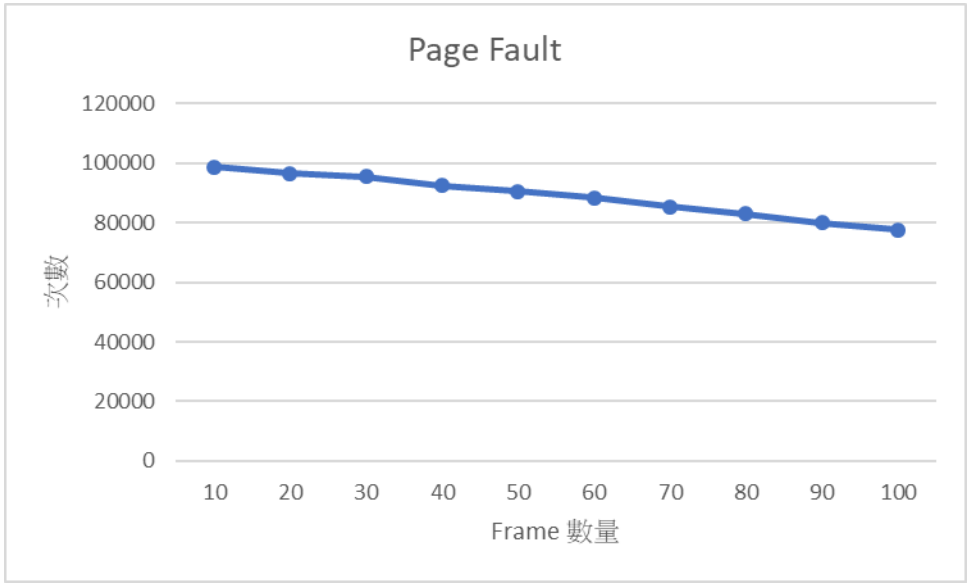

Page Fault
120000
100000
80000
60000
40000
20000
0
次數
10
20
30
40
50
60
70
80
90
100
Frame 次數

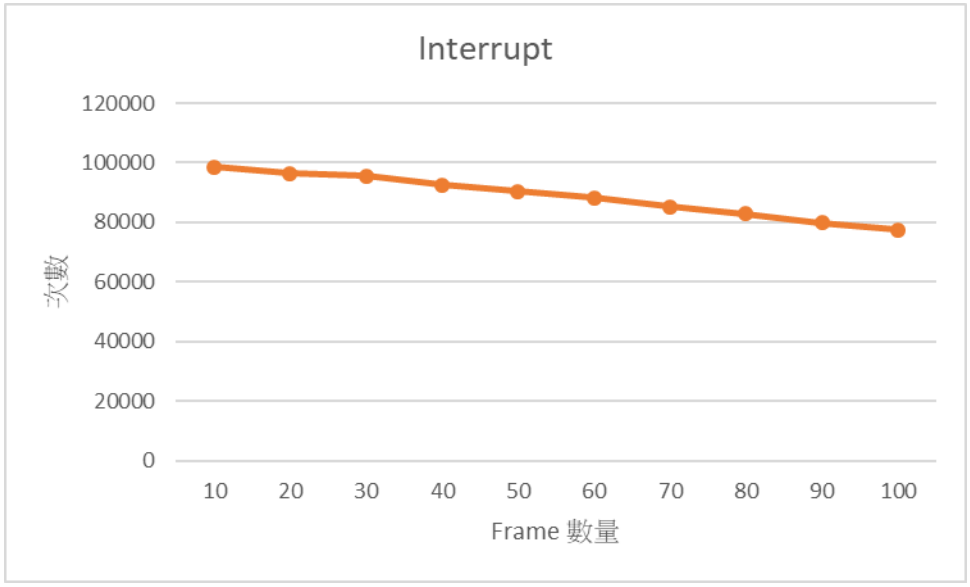

Interrupt
120000
100000
80000
60000
40000
20000
0
次數
10
20
30
40
50
60
70
80
90
100
Frame 數量

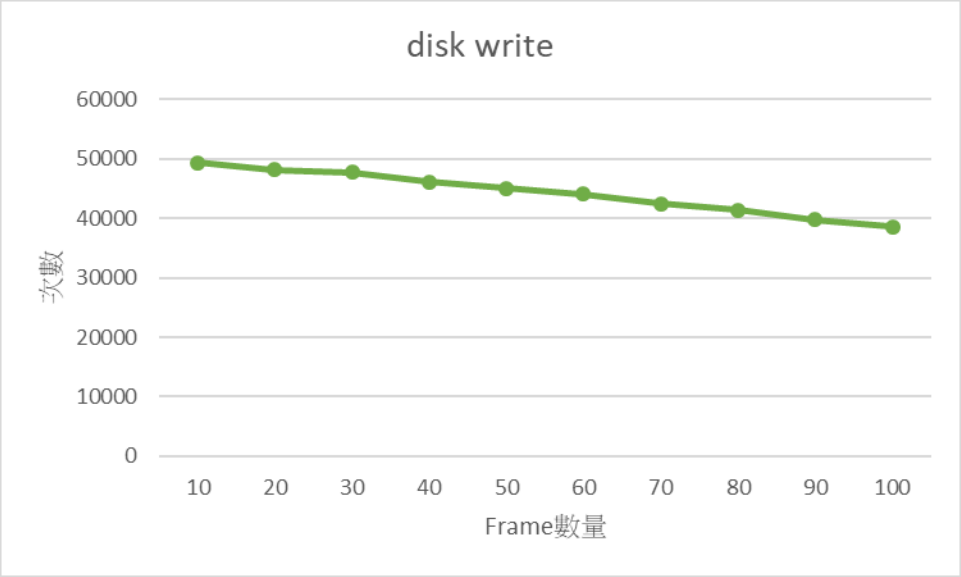

disk write
60000
50000
40000
30000
20000
10000
0
次數
10
20
30
40
50
60
70
80
90
100
Frame 次數

	Page Fault	Interrupt	disk write
10	97964	97954	48769
20	96030	96010	47785
30	94060	94030	46808
40	92103	92063	45790
50	89903	89853	44677
60	88048	87988	43728
70	86082	86012	42753
80	83974	83894	41659
90	82121	82031	40747
100	80000	79900	39571

>esc_Locality_rs.py

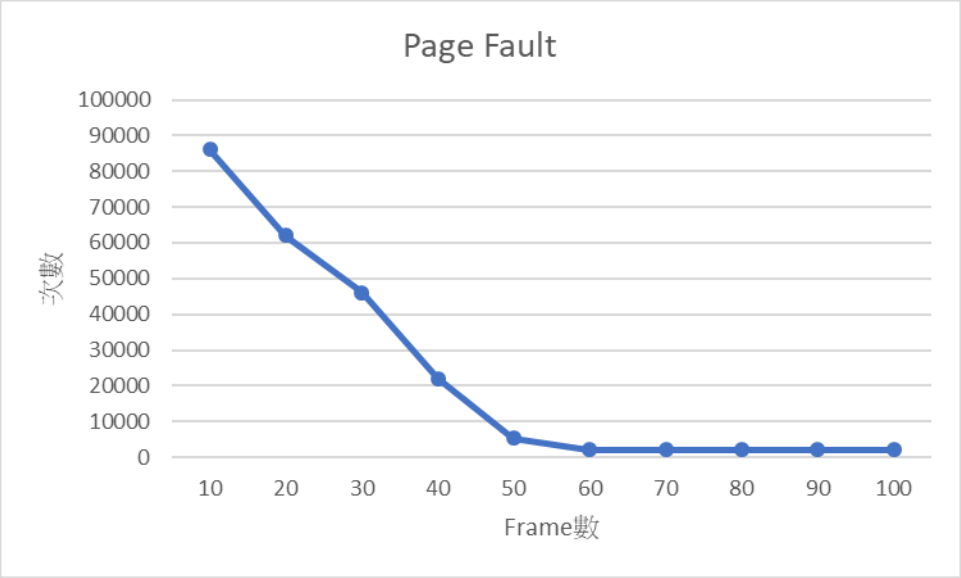


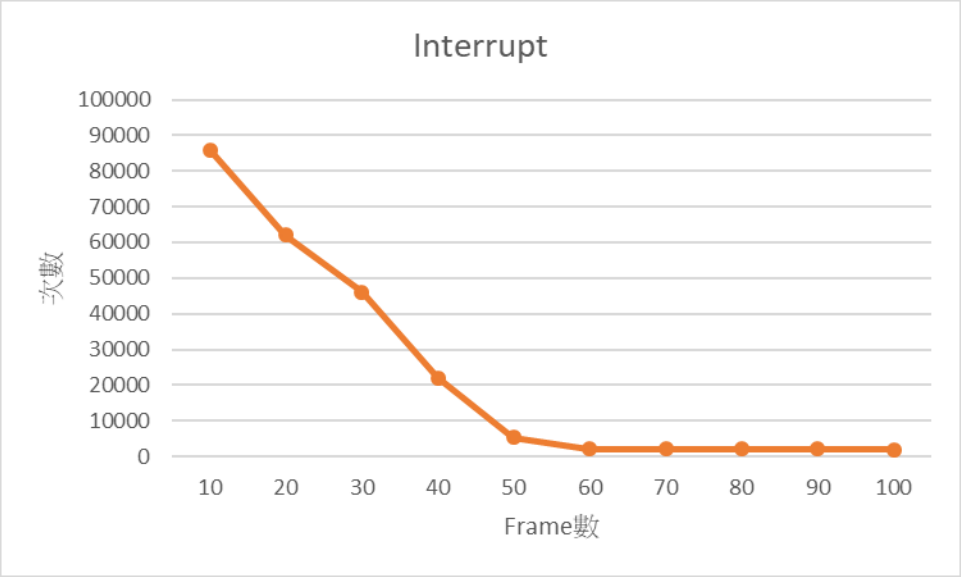


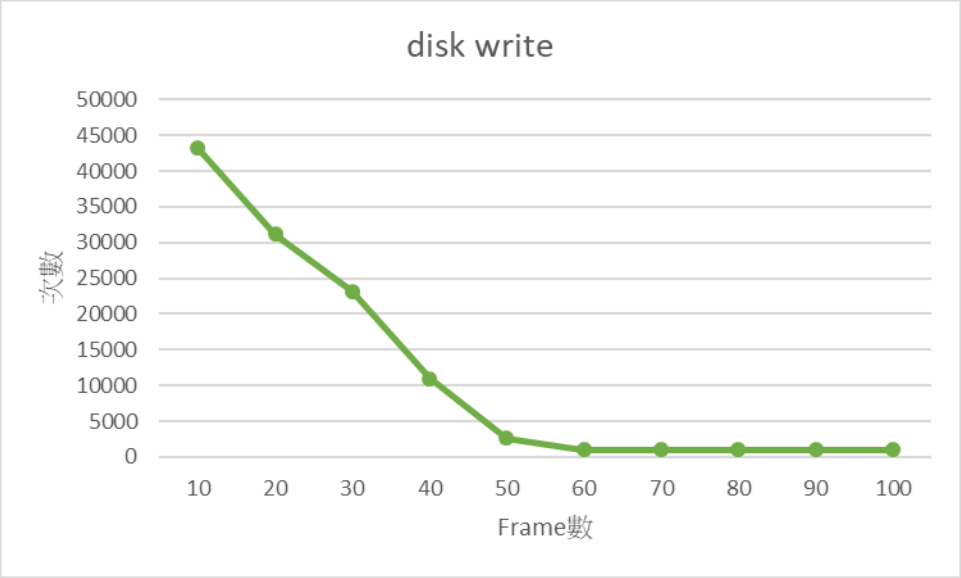


	Page Fault	Interrupt	disk write
10	98653	98643	49295
20	96527	96507	48191
30	95558	95528	47711
40	92486	92446	46141
50	90391	90341	45055
60	88393	88333	44065
70	85304	85234	42488
80	82975	82895	41326
90	79918	79828	39773
100	77623	77523	38617

>esc_custom.py







	Page Fault	Interrupt	disk write
10	86007	85997	43153
20	62019	61999	31162
30	46029	45999	23178
40	22046	22006	10989
50	5295	5245	2579
60	2059	1999	998
70	2069	1999	994
80	2079	1999	987
90	2089	1999	982
100	2098	1998	976

4. Custom Algorithm : LRU_Pro

第一優先 被替換掉: 在 frame 累積出現次數最少。

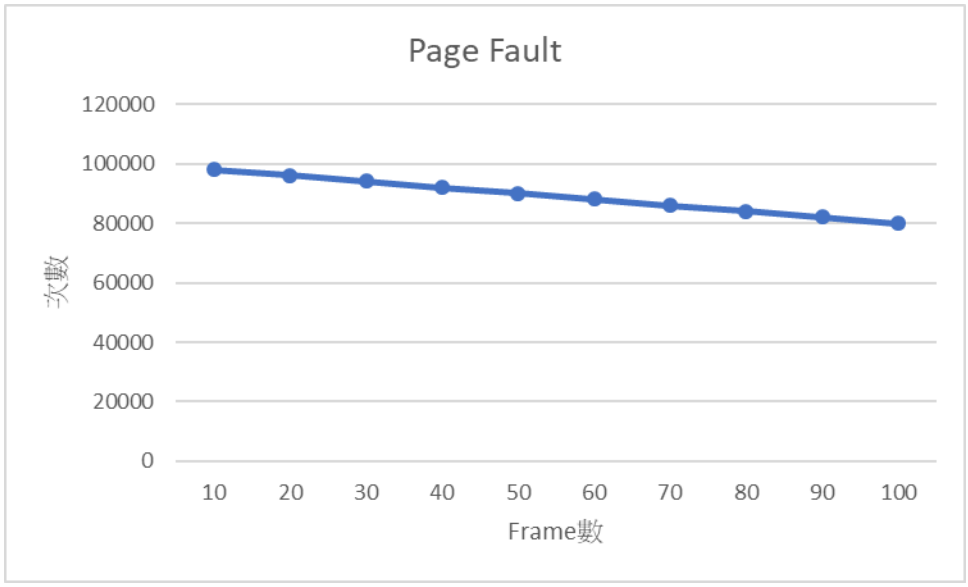
當 累積次數最小是一樣的話，進入第二優先。

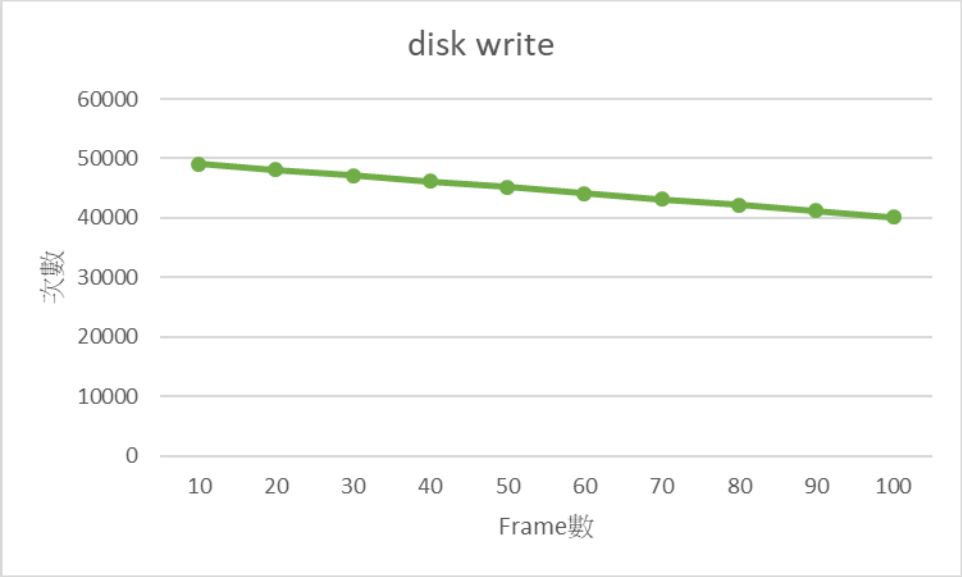
第二優先 被替換掉: 最近沒被使用到(很久沒被用到)。

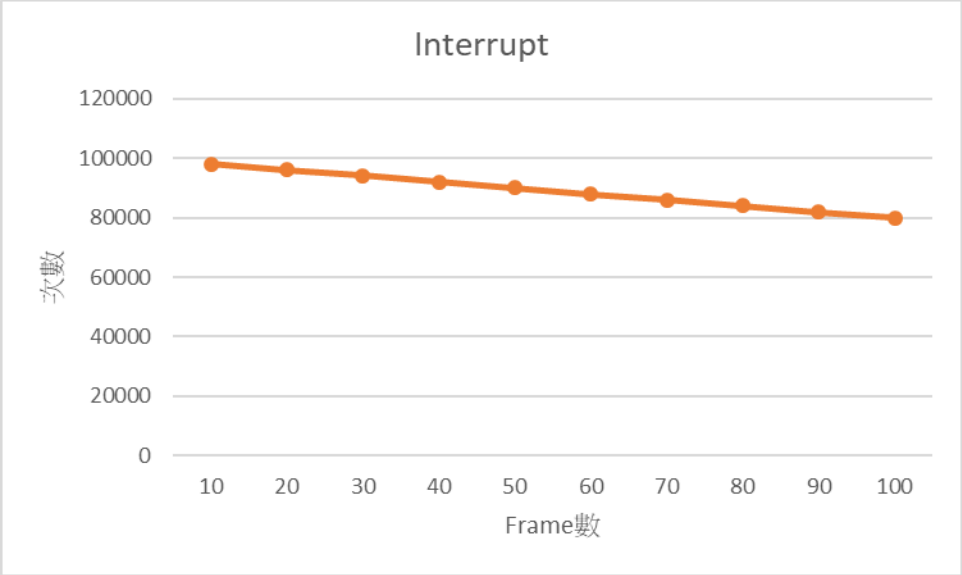
```
for z in range(10):
    fn = input("輸入 frame 數: ")
    fn = int(fn)
    page_fault = 0
    dw = 0
    interrupt = 0

    frame = [] # 記憶體 狀況
    frame_record = [] # 紀錄出現次數

    index = 0 # 到哪個 reference
    for i in range(fn):

        frame.append(rs[index])
        frame_record.append(1) # 從 1 開始，計算次數
        index = index + 1
        page_fault = page_fault + 1

    print(page_fault) # 顯示目前 page fault

    while index < len(rs):
        if rs[index] in frame:
            # 次數 + 1
            frame_record[frame.index(rs[index])] = frame_record[frame.index(rs
[index])] + 1
            # 變更 累積次數
            tmp1=frame_record[frame.index(rs[index])]
            del frame_record[frame.index(rs[index])]
            frame_record.append(tmp1)

            # 變更 記憶體 狀況
            # 新進 塞最後
            tmp2 = frame[frame.index(rs[index])]
            del frame[frame.index(rs[index])]
            frame.append(tmp2)

        else:

            min_value = min(frame_record) # 找出次數最少的數量

            # 找離目前 reference page 最近的 reference page
            # 如果 dirty bit 為 1 , 則 disk write++
            for k in range(index-1,0,-1):
                if rs[k] == frame[frame_record.index(min_value)]:
```

```
                    if dw_set[k] == 1 :
                        dw = dw + 1
                    break
            # 替換，並新增
            frame.remove(frame[frame_record.index(min_value)])
            frame.append(rs[index])

            del frame_record[frame_record.index(min_value)]
            frame_record.append(1) # 該 frame 次數 重設為 1

            interrupt = interrupt + 1
            page_fault = page_fault + 1

        index = index + 1
    print(page_fault)
    print(interrupt)
    print(dw)
```

>LRU_Pro_Random.py

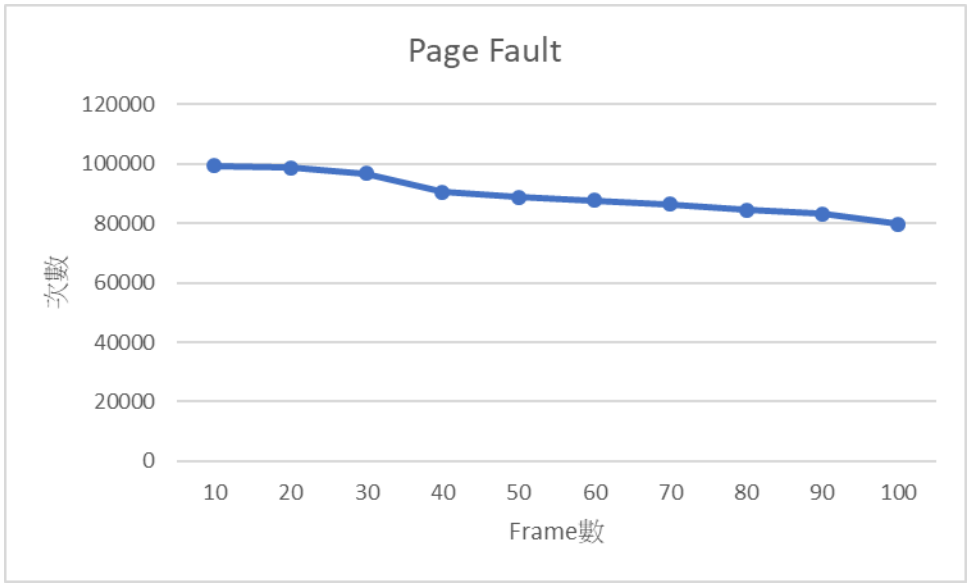


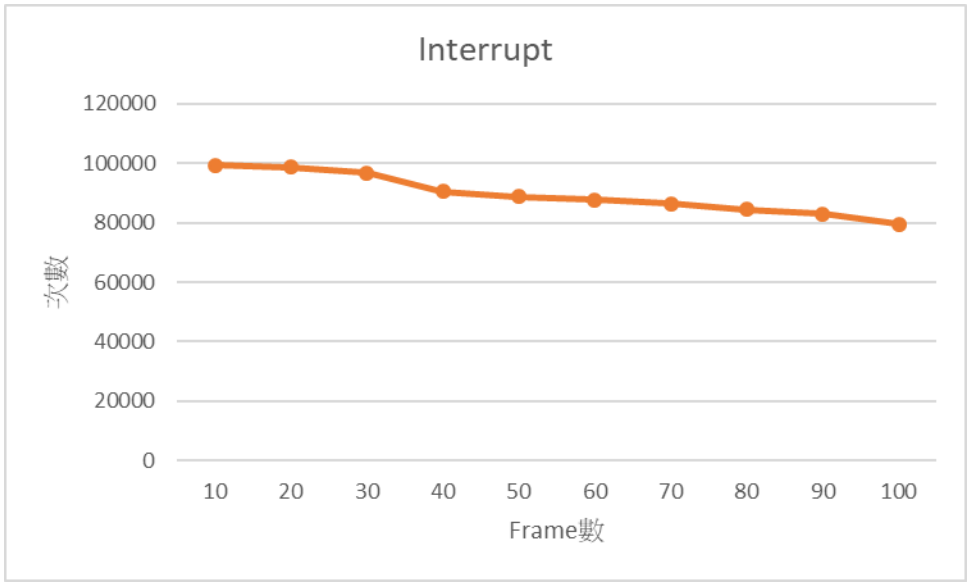


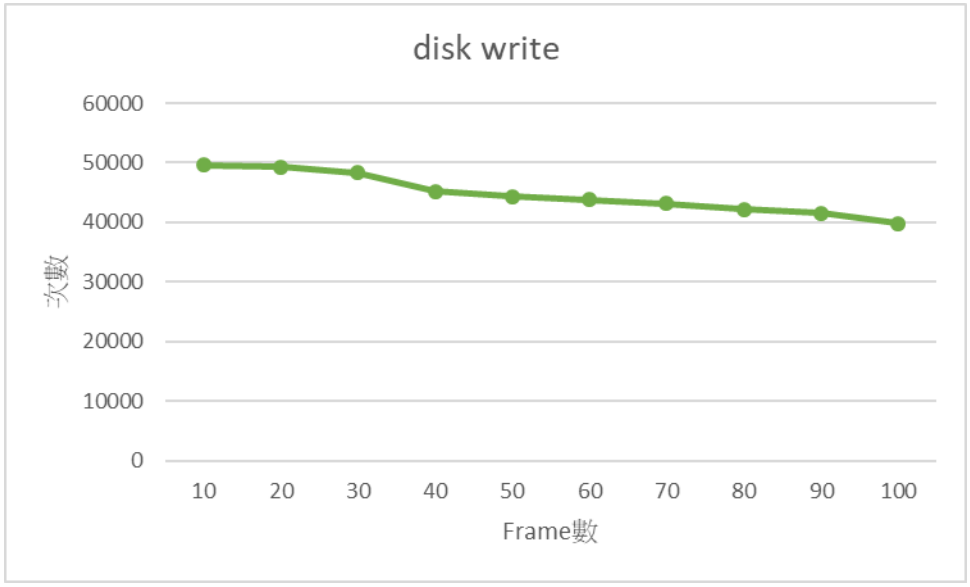


	Page Fault	Interrupt	disk write
10	98004	97994	49051
20	96086	96066	48126
30	94087	94057	47154
40	92052	92012	46146
50	90028	89978	45149
60	88054	87994	44156
70	86039	85969	43163
80	84058	83978	42202
90	82053	81963	41180
100	80025	79925	40161

>LRU_Pro_Locality.py

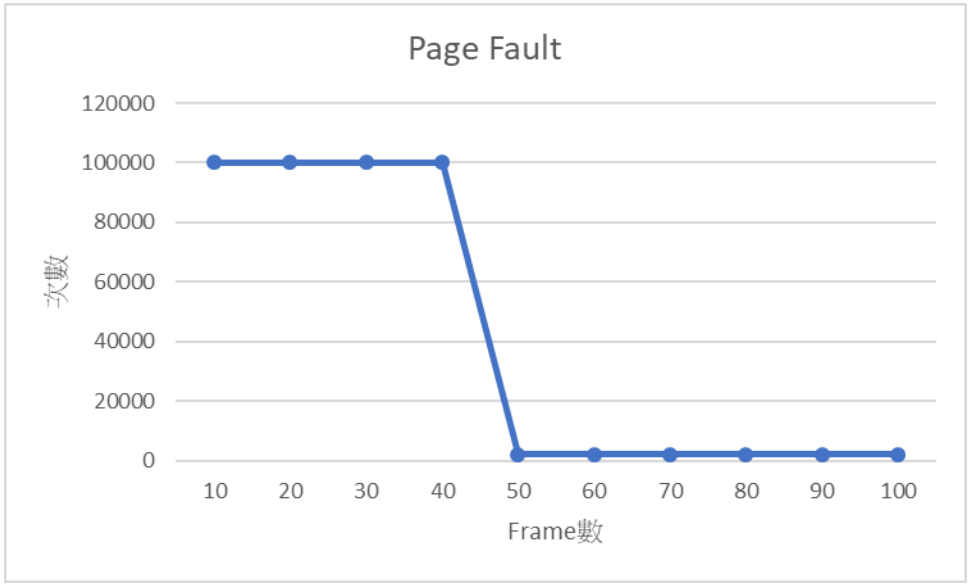

Page Fault
120000
100000
80000
60000
40000
20000
0
次數
10
20
30
40
50
60
70
80
90
100
Frame數

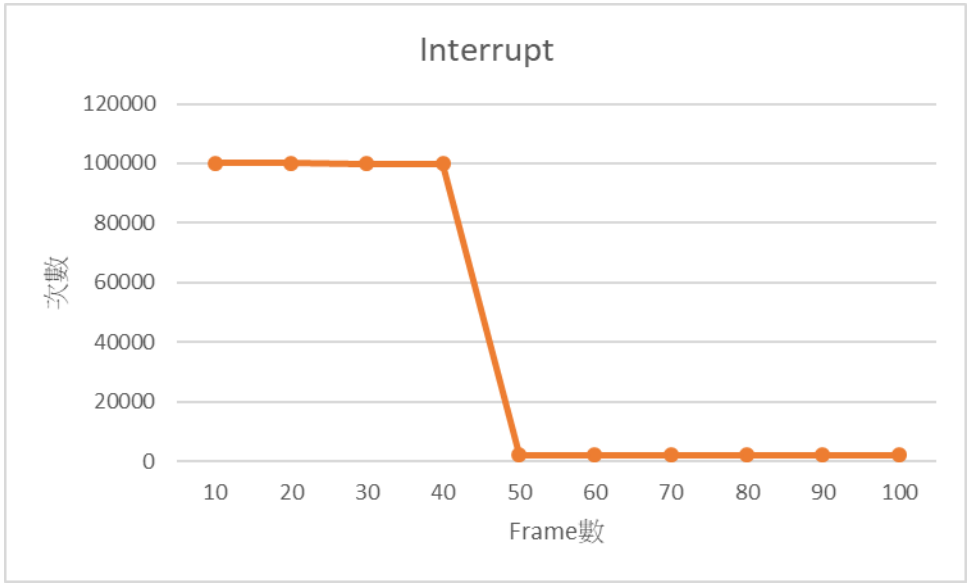

Interrupt
120000
100000
80000
60000
40000
20000
0
次數
10
20
30
40
50
60
70
80
90
100
Frame數

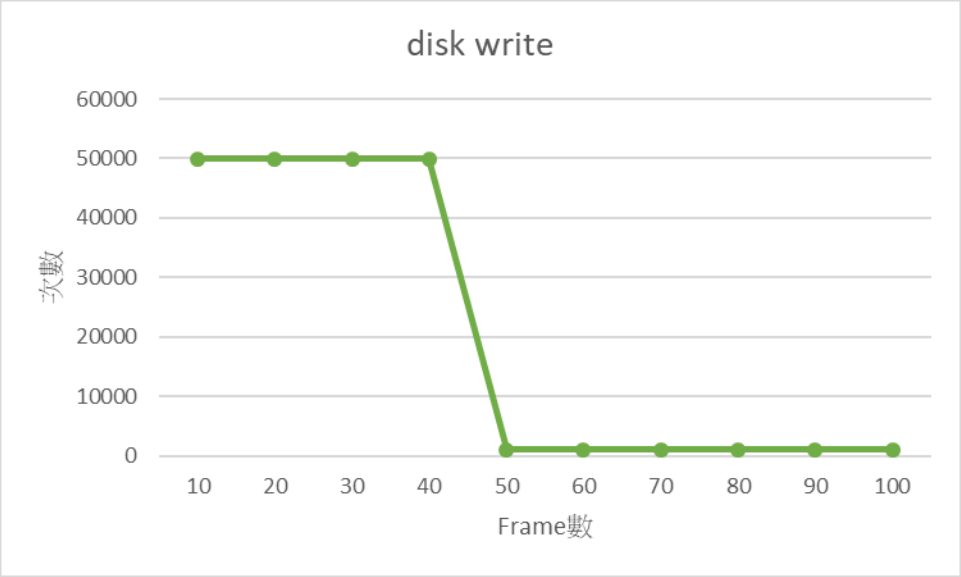

disk write
60000
50000
40000
30000
20000
10000
0
次數
10
20
30
40
50
60
70
80
90
100
Frame數

	Page Fault	Interrupt	disk write
10	99277	99267	49605
20	98782	98762	49341
30	96754	96724	48312
40	90473	90433	45208
50	88844	88794	44353
60	87701	87641	43822
70	86465	86395	43149
80	84479	84399	42190
90	83146	83056	41546
100	79676	79576	39793

>LRU_pro_custom.py



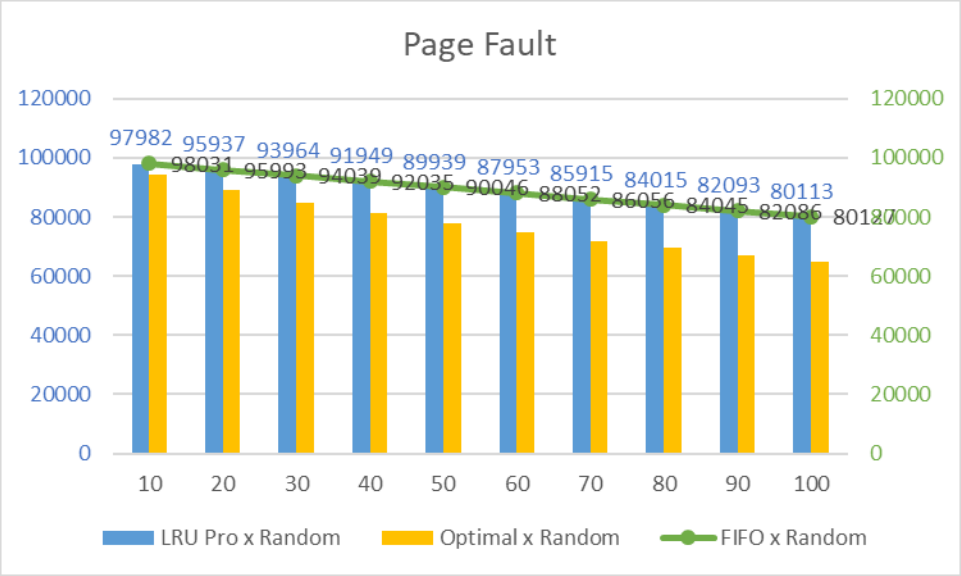




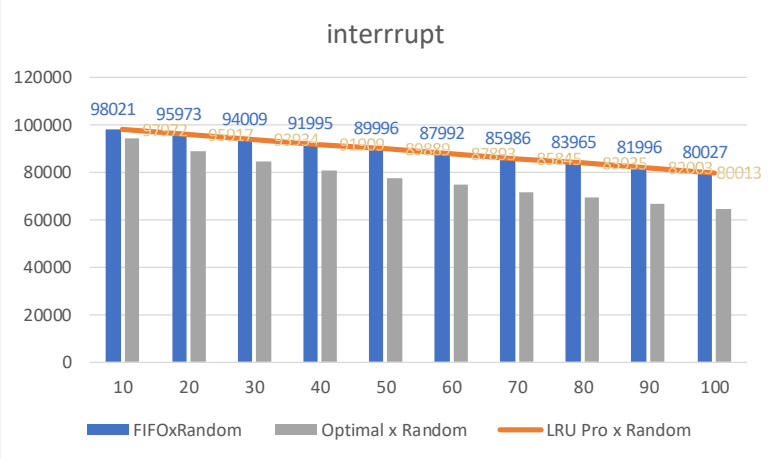
	Page Fault	Interrupt	disk write
10	100000	99990	49963
20	100000	99980	49960
30	100000	99970	49955
40	100000	99960	49952
50	2049	1999	1002
60	2059	1999	1002
70	2069	1999	1002
80	2079	1999	1002
90	2089	1999	1002
100	2098	1998	1002

FIFO vs LRU Pro (自訂演算法) vs Optimal : Random

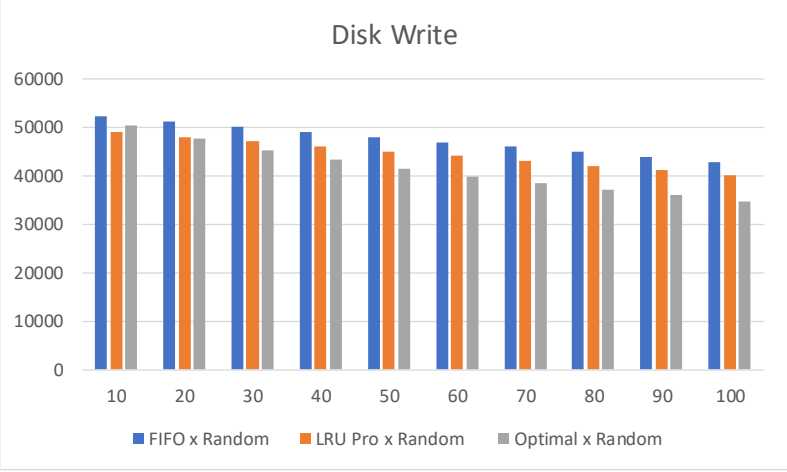
[Page Fault 分析] LRU Pro (自訂演算法) 略勝於 FIFO。



[Interrupt 分析] LRU Pro (自訂演算法) 險勝於 FIFO。

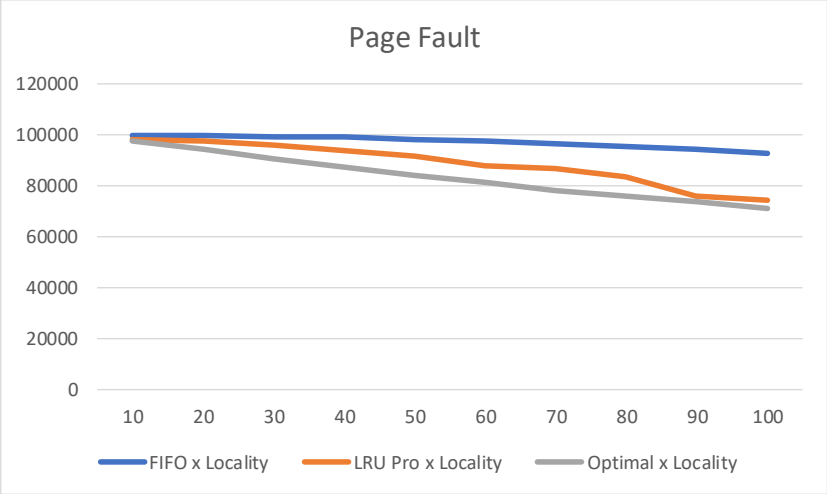


[disk write 分析]



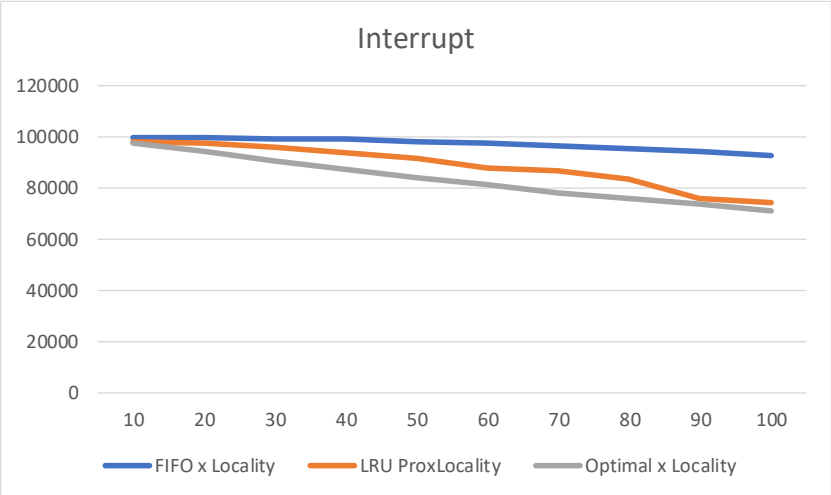
FIFO vs LRU Pro (自訂演算法) vs Optimal :Locality

[分析 Page Fault] LRU Pro (自訂演算法) 隨著 Frame 數 ，拉開與 FIFO 的差距。

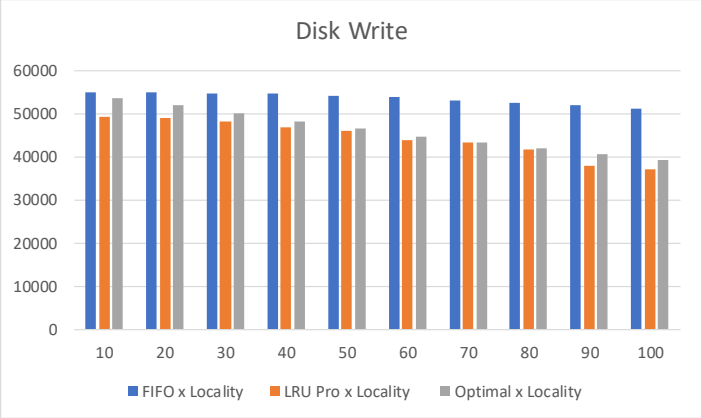


[Interrupt 分析] LRU Pro (自訂演算法) 隨著 Frame 數 ，拉開與 FIFO 的差距。

現象和 Page Fault 差不多。



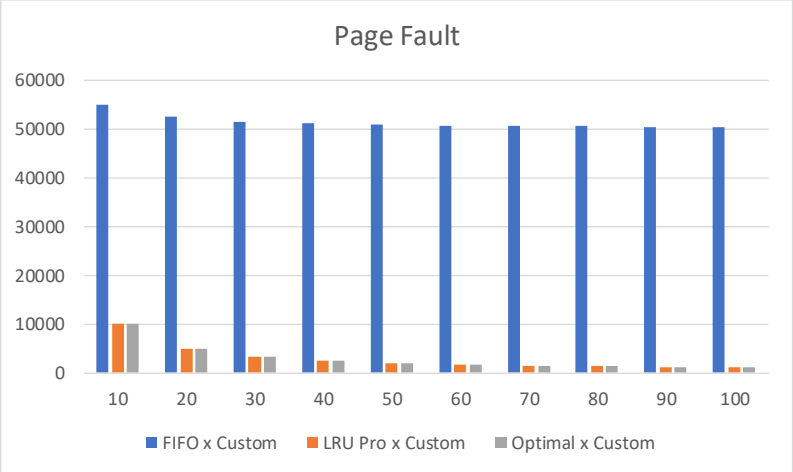
[disk write 分析]



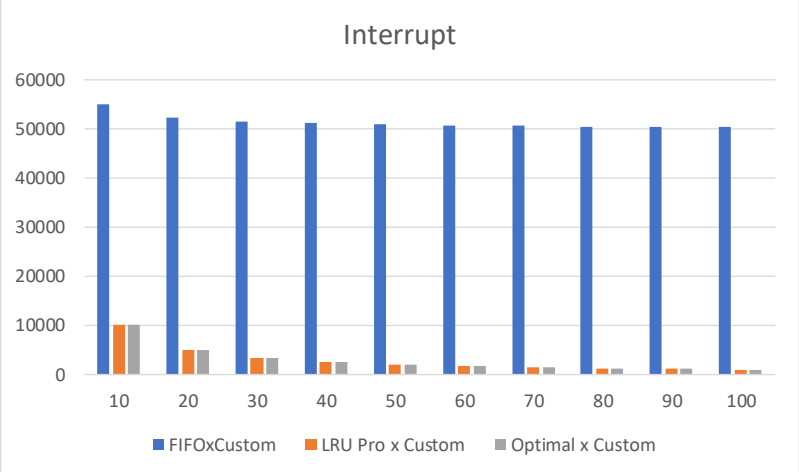
FIFO vs LRU Pro (自訂演算法) vs Optimal :Custom

[分析 Page Fault] 專為 LRU Pro 設計 ，Optimal == LRU Pro

[說明] Custom 是專為 LRU page 設計的，雖然 LRU Pro 不能像 Optimal 預知未來，但以優先 1 的篩選方式(次數最多不換)，讓將要替換的最後一個 page，次數最少，所以在執行時間上，LRU Pro 會比 Optimal 還要來的快，同時達到 Optimal 的效果，在 Custom Reference String 的 Case 上。



[Interrupt 分析]



[disk write 分析]