**รายงาน**

เรื่อง Page replacement algorithms

**เสนอ**

[รศ.ดร.นริศรา เอี่ยมคณิตชาติ](https://cpe.eng.cmu.ac.th/2013/academic-view.php?academic_id=10)

**จัดทำโดย**

นาย ธนาคม หัสแดง

590610624

รายงานนี้เป็นส่วนหนึ่งของวิชา Operating System

ภาคเรียนที่ 2 ปีการศึกษา 2561

มหาวิทยาลัยเชียงใหม่

***Program***

*• แนวคิดในการพัฒนาแต่ละ algorithm*

*- FIFO ใช้ stack โดย process ไหนที่เข้าก่อน ก็จะเปลี่ยนค่าใน page ก่อน ตามลำดับของการเข้ามาใน stack*

*- Optical จะเป็นการ ดูว่า page ไหน ไม่ได้ถูกใช้งานในเวลาที่นานที่สุด เมื่อได้แล้วจึงนำ process นั้นไป เปลี่ยนค่า*

*- LRU จะเป็นการดูว่า process ใด ที่มีลำดับห่างจาก process ปัจจุบันมากที่สุด*

***Code***

import java.io.IOException;

import java.util.Random;

import java.io.BufferedWriter;

import java.io.FileWriter;

import java.lang.Math;

public class main {

*//////////////// ค่า คงตัวที่จำเป็นต้องใช้ในการคำนวณ /////////*

public static final int expire\_time = 10;

public static final int process\_size = 1200;

public static final int page\_size = 80;

public static final int process\_seq = 120;

public static int value\_fifo = 0, value\_opt = 0, value\_lru = 0;

*//////////////// function main ////////////*

public static void main(String *args*[]) throws IOException {

*///////////////////////// สร้าง ไฟล์มาเก็บค่า เพื่อในไปใช้ในการดูผลต่อไป //////////*

BufferedWriter stream\_out = new BufferedWriter(new FileWriter("output.csv"));

stream\_out.write("# of Process," + process\_size + "\n");

stream\_out.write("# of Page," + page\_size + "\n");

stream\_out.write("Page fault occurrence\n");

stream\_out.write("Round,FIFO,OPT,LRU\n");

int sum\_fifo = 0, sum\_opt = 0, sum\_lru = 0;

Random rand = new Random();

*////////// วลูปตามจำนวณ expire time ที่เราต้องการ ////////*

for (int count = 0; count < expire\_time; count++) {

value\_fifo = 0;

value\_opt = 0;

value\_lru = 0;

*////////// สร้าง Array process และ page มา //////*

int[] p = new int[process\_size];

int[] page = new int[page\_size];

int[] def\_p = new int[process\_size];

*////////////// กรณี กำหนด p /////////////////*

*//int[] def\_p\_new = {1,2,3,4,5,1,2,3,4,5,4,5};*

*/\*for (int i = 0 ; i < 12 ; i++ ){*

*def\_p[i] = def\_p\_new[i];*

*} \*/*

*/////////// สุ่มลำดับ โพรเซส ///////////////*

for (int i = 0; i < process\_size ; i++) {

int x = Math.abs(rand.nextInt()) % process\_seq;

while (x == def\_p[i]) {

x = Math.abs(rand.nextInt()) % process\_seq;

}

def\_p[i] = x;

}

*/////////// Reset ค่า page และ กำหนดค่า p ใหม่ //////////////*

page = Reset(page);

p = Set\_p(def\_p);

*///////// เข้า function ทั้ง 3 algorithm ////////////*

FIFO(page, p);

Optimal(page, p);

LRU(page, p);

*//////////// นำค่า ของ algorithm แต่ละอันในแต่ละรอบ มาบวกจากค่าเดิม /////////*

sum\_fifo += value\_fifo;

sum\_opt += value\_opt;

sum\_lru += value\_lru;

stream\_out.write(count + 1 + "," + value\_fifo + "," + value\_opt + "," + value\_lru + "\n");

}

stream\_out.write("AVG : ," + sum\_fifo / expire\_time + "," + sum\_opt / expire\_time + "," + sum\_lru / expire\_time);

stream\_out.close();

}

*////////////// function reset ค่า page ให้เป็น -1 ทั้งหมด ///////////*

public static int[] Reset(int[] *page*) {

for (int i = 0; i < page\_size; i++) {

page[i] = -1;

}

return page;

}

*///////////// ฟังก์ชั่น กำหนดค่า p ให้มีค่าเปรียบเหมือน def\_f //////////*

public static int[] Set\_p(int[] *def\_p*) {

int[] k = new int[process\_size];

for (int i = 0; i < process\_size ; i++) {

k[i] = def\_p[i];

}

return k;

}

*//////////// function FIFO ///////////////////*

public static void FIFO(int[] *page*, int[] *p*) {

int count = 0;

*//////////// loop ตามจำนวน process ////////////*

for (int i = 0; i < process\_size; i++) {

if (count >= page\_size){

count = 0;

}

*////////////// เช็คว่ามี process อยู่ใน page นั้นหรือไม่ ///////////*

int checknum = Find\_page(page, p[i]);

*////////////// ถ้าไม่เจอ ให้เพิ่มค่า value ของ fifo และ ปรับค่า page ตาม process ใหม่ /////////*

if ((page[count] < 0 && checknum == -1) || checknum == -1) {

page[count] = p[i];

count++;

value\_fifo++;

}

}

System.out.println("FIFO page\_falut : " + value\_fifo);

}

*//////////////// function Optimal ///////////////*

public static void Optimal(int[] *page*, int[] *p*) {

*///////////// กำหนดค่าต่างๆ ///////////////*

int max = -1;

int[] x = new int[page\_size];

int idx = -1;

*////////////// กำหนดค่า page นั้นๆ ให้เท่ากับ process ที่ตรงกัน /////////*

for (int i = 0; i < page\_size; i++) {

page[i] = p[i];

value\_opt++;

}

*//////// loop ตั้งแต่ลำดับ page ไปจนจบ ลำดับ process /////////*

for (int i = page\_size; i < process\_size; i++) {

max = -1;

idx = -1;

for (int j = 0; j < page\_size; j++) {

x[j] = -1;

}

*////////////// เช็คว่ามี process อยู่ใน page นั้นหรือไม่ ///////////*

if (Find\_page(page, p[i]) < 0) {

*///////// loop จากลำดับของ page ไปจนจบ ลำดับ process เพื่อหา page ไหนถูกเรียกเป็นลำดับสุดท้าย ///////*

for (int k = i; k < process\_size ; k++) {

int count = Find\_page(page, p[k]);

if (k > idx && count >= 0) {

if (x[count] == -1) {

idx = k;

max = count;

x[count] = count;

}

}

}

for (int k = 0; k < page\_size; k++) {

if (x[k] < 0) {

max = k;

}

}

if (max == 1) {

max = 0;

}

page[max] = p[i];

value\_opt++;

}

}

System.out.println("Optimal page\_fault : " + value\_opt);

}

*/////////// function LRU ///////////////*

public static void LRU(int[] *page*, int[] *p*) {

*/////////////////// กำหนดค่าเริ่มต้น /////////////////*

int idx = 0;

int checknum = 0;

int[] history = new int[page\_size];

*/////////////// กำหนดค่า ให้ page นั้นๆ เท่ากับ process และ กำหนดค่า history ////////*

for (int i = 0; i < page\_size; i++) {

history[i] = 0;

page[i] = p[i];

history[i] += i;

value\_lru++;

}

*///////////// loop ตั้งแต่ ลำดับ page size จนจบ process size //////////////*

for (int i = page\_size; i < process\_size; i++) {

*////////////// เช็คว่ามี process อยู่ใน page นั้นหรือไม่ ///////////*

checknum = Find\_page(page, p[i]);

*///////////// ถ้า มี ให้กำหดค่า history เป็นค่าลำดับใหม่ //////////*

if (checknum >= 0) {

history[checknum] = i;

} else {

*//////////////////// ถ้าไม่อยู่ ให้วนอีกรอบ ตั้งแต่ 0 ถึงลำดับ page size ////////*

for (int j = 0; j < page\_size; j++) {

int min = history[j];

for (int k = 0; k < page\_size; k++) {

*///////////////// ค้นหา page ที่เห็บ process ใหม่ด้วยการ หา min ///////////*

if (history[k] < min) {

idx = k;

min = history[k];

}

}

}

page[idx] = p[i];

history[idx] = i;

value\_lru++;

}

}

System.out.println("LRU page\_falut : " + value\_lru + "\n");

}

*////////////////// function ดูว่า มี process อยุ่ใน page นี้หรือไม่ /////////*

public static int Find\_page(int[] *page*, int *p*) {

for (int i = 0; i < page\_size; i++) {

if (page[i] == p)

return i;

}

*//////////// ไม่มี return -1 /////////*

return -1;

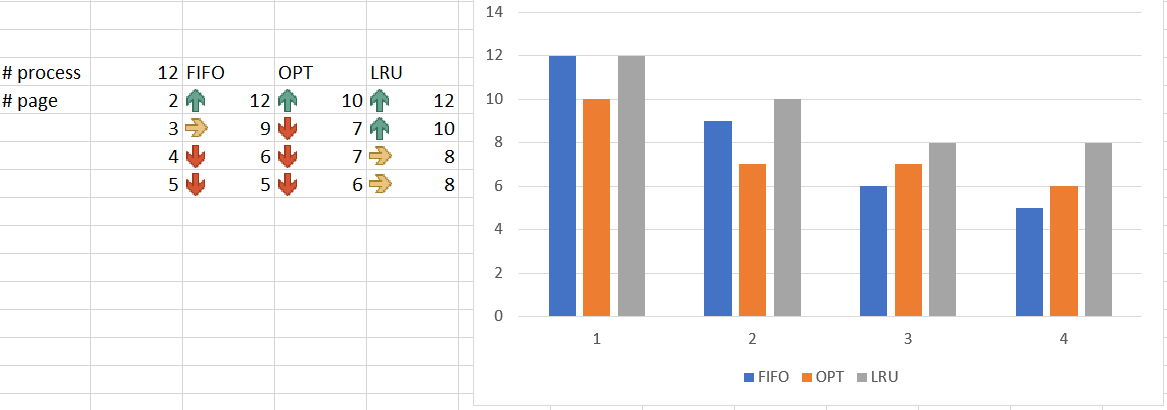
}

}

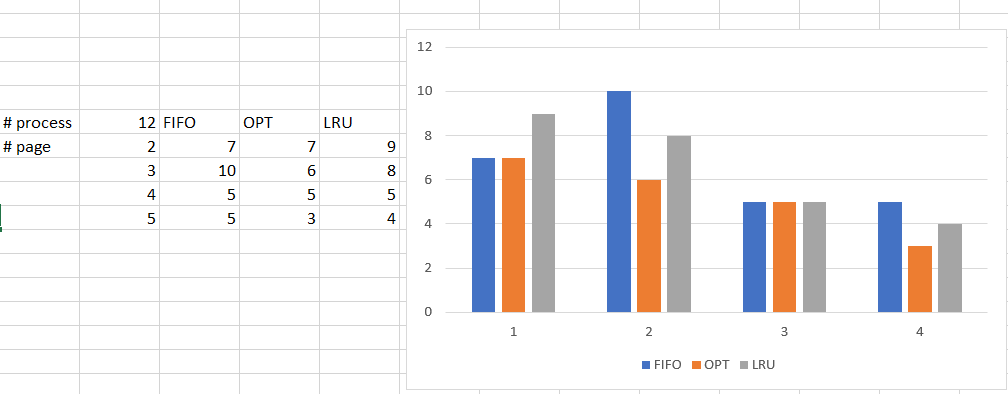
***สมมติฐานการทดลอง***

*1) Belady’s anomaly ทดสอบว่าเมื่อเพิ่มจำนวน page จะทำให้เกิดการเพิ่มของ page fault ในprocess นั้นๆ*

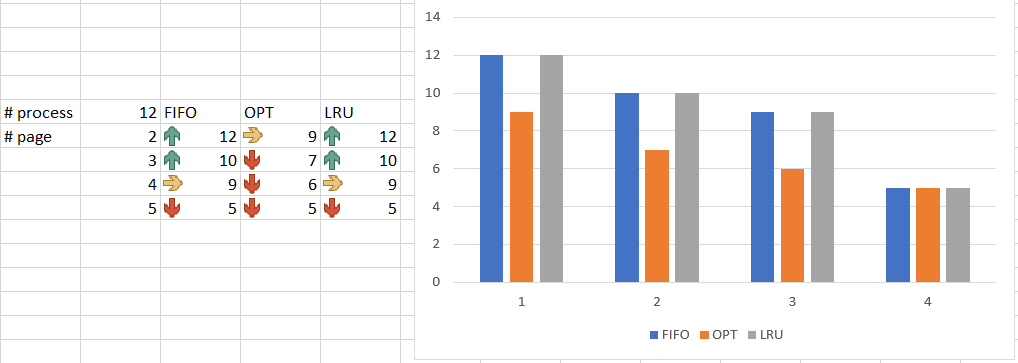
*Reference string = {1,2,3,4,5,1,2,3,4,5,4,5}*

**

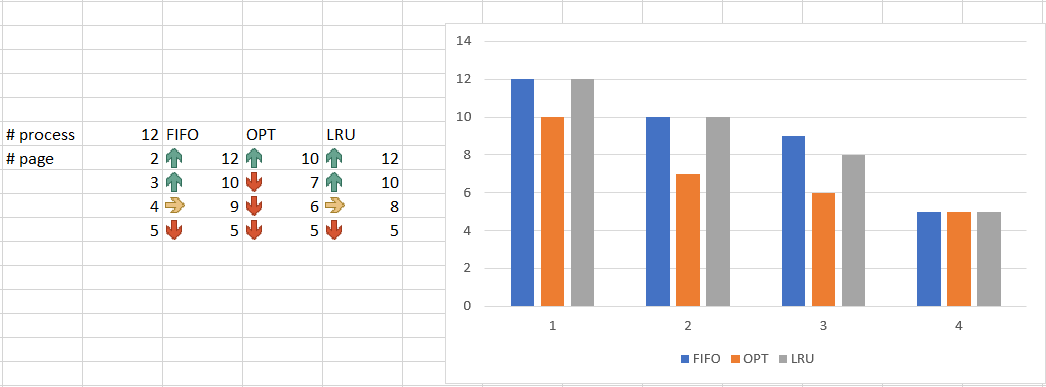
*Reference string = {0,1,2,3,1,2,4,0,1,2,3,4}*

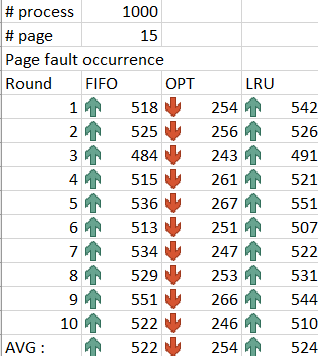
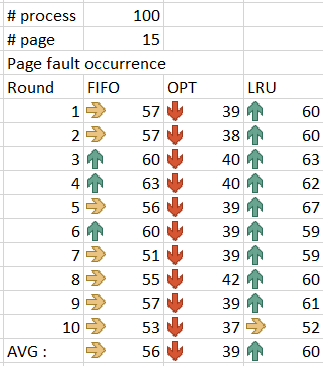
**

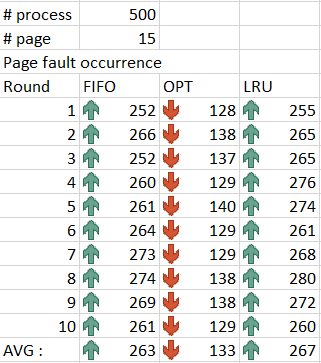
*Random reference string = {2,1,2,0,1,4,2,0,1,4,0,3}*

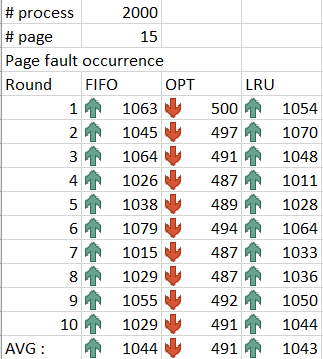
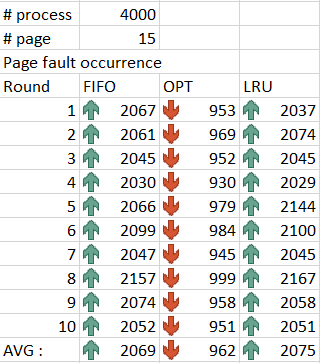
**

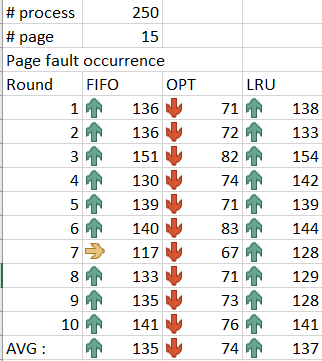
*Random reference string = { 3,2,1,4,5,1,5,3,4,2,5,1}*

**

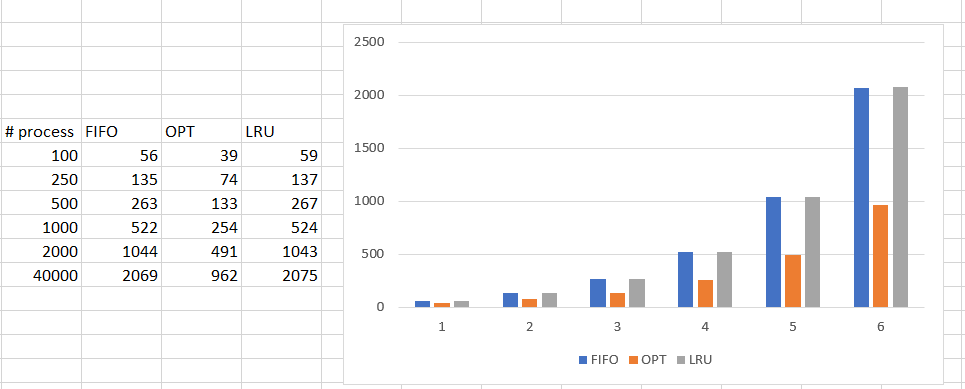
*2)กรณีที่เพิ่มจำนวน process แต่ไม่เพิ่มจำนวน page จะทำให้โอกาสการเกิด page fault ของ OPT จะน้อยกว่า LRU และ FIFO*

**

**

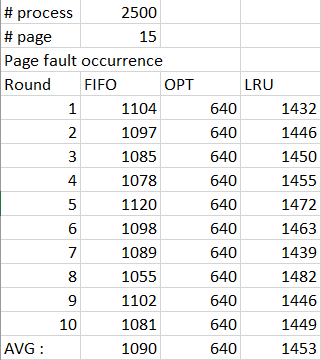
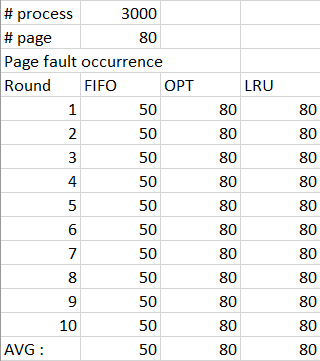
**

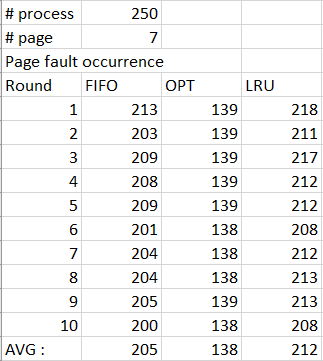
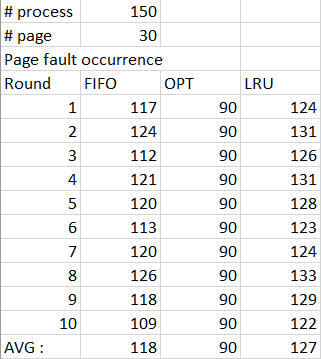
*นำค่าเฉลี่ยๆมาเปรียบเทียบ*

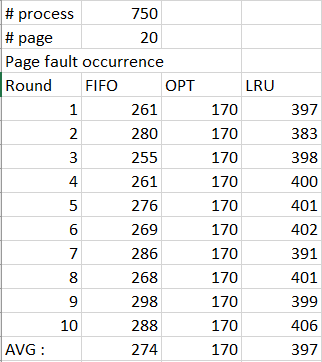
**

*3)ค่า Random reference string เมื่อมาสุ่มลำดับนั้น ผลที่ออก จะมีค่าที่ออกจากอัลกอริทึ่ม OPT มากที่สุดในทุก process และ page เสมอ*

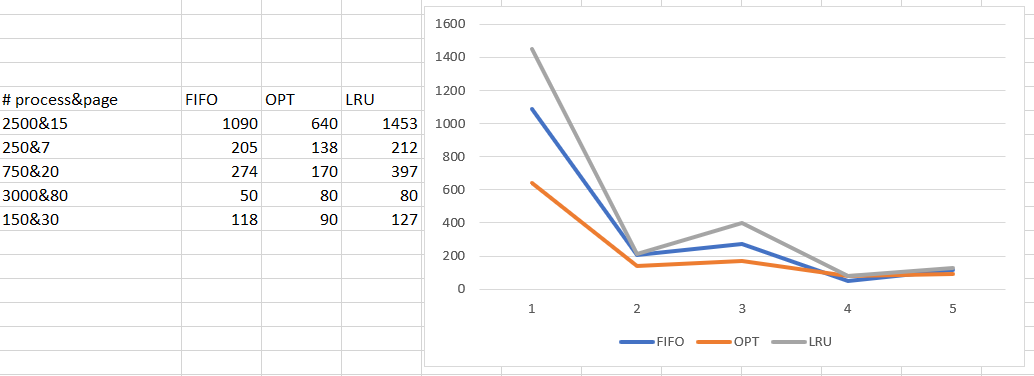
*4) ค่า Random reference string เมื่อมาสุ่มลำดับ ผลที่ออกของทั้ง 3 อัลกอริทึ่มจะมีค่าใกล้เคียงกันเสมอ*

**

**

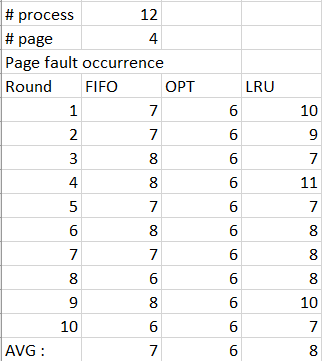
**

*จะเห็นว่า ค่าของ อัลกอริทึ่มที่ออกมา จะใกล้เคียงกันเสมอ และ ค่า LRU จะมีค่ามากกว่า อีก 2 อัลกอริทึ่มเสมอ*

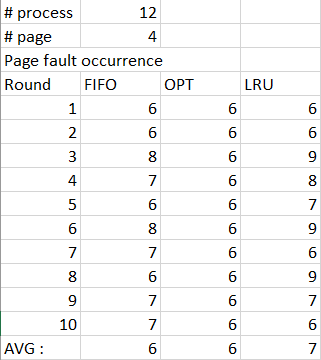
**

*5) ไม่ว่าเราจะจัดเรียงข้อมูลใหม่เพียงใด ถ้าเป็นข้อมูลเดิมจะทำให้ค่า OPT มีค่าเท่าเดิมเสมอ*

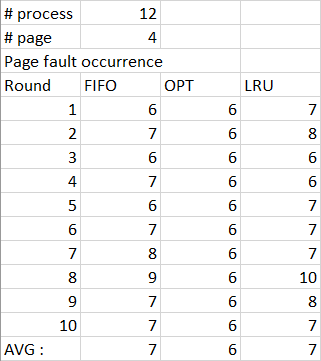
*Random reference string = {1,2,1,2,3,1,2,1,2,3,3};*

**

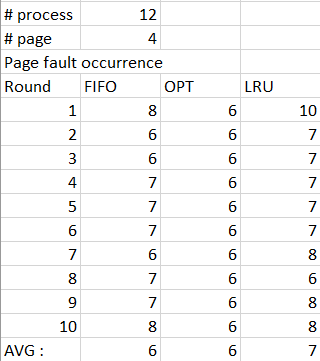
*Random reference string = {1,2,1,2,3,3,1,2,1,2,3};*

**

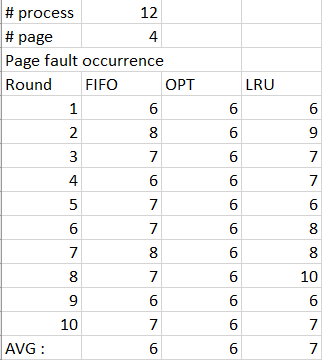
*Random reference string = {2,1,2,1,2,3,3,3,1,2,1};*

**

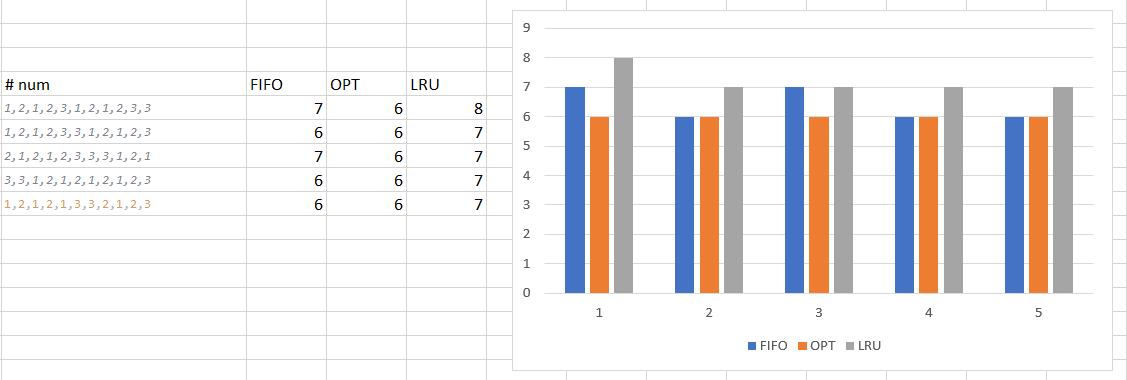
*Random reference string = {3,3,1,2,1,2,1,2,1,2,3};*

**

*Random reference string = {1,2,1,2,1,3,3,2,1,2,3};*

**

*เมื่อนำค่าเฉลี่ยของแต่ละแบบมาเปรียบเทียบ จะเห็นว่า OPT มีค่าเท่ากันทุกการทดลอง*

**

***รูปแบบ Reference string ในการทดลอง***

* *สุ่มลำดับทั้งหมด โดย จะรวมถึงการสุ่มค่าในลำดับด้วย*
* *สุ่มลำดับจากเซตที่เรากำหนดมา โดยเซตที่เรากำหนดจะ จัดเป็นเซตที่เท่ากัน และในแต่ละข้อมูลจะมีลำดับของข้อมูลครบทุกค่า*

***ประเมินผลการทดลองแต่ละ algorithm***

* *FIFO นั้นจำนวนการเกิด page fault จะใกล้เคียงกับ LRU ซึ่งทั้ง 2 นั้นจะแย่กว่า OPT ที่เร็วกว่า และเป็นเพียง อัลกอริทึ่มเดียวที่อาจเกิด Belady’s anomaly และเป็น อัลกอริทึ่มที่เข้าใจง่ายมากที่สุด*
* *OPT นั้นมีโอกาสการเกิด page fault น้อยมากโดย ใน reference string ส่วนมากจะให้ ผลลัพธ์ที่ดีกว่าทั้ง 2 แบบที่เหลืออย่างมาก และข้อดีคือจะไม่เกิด Belady’s anomaly อย่างแน่นอน*
* *LRU คล้ายกับ FIFO แต่มีความยากที่มากกว่า FIFO มาก และข้อดีคือจะไม่เกิด Belady’s anomaly อย่างแน่นอน*

***สรุปผลแต่ละการทดลอง***

***จากสมตติฐานที่ 1*** *เมื่อทดสอบ Belady’s anomal จะพบว่ามีโอกาสเกิดได้ใน reference string ได้ในหลายๆรูปแบบแต่ไม่ทุกแบบ 0,1,2,3,1,2,4,0,1,2,3,4 ทำให้เกิด กราฟที่ มีการ เพิ่มขึ้นของค่า page fault และจะเกิดเพียงเฉพาะกับ FIFO เท่านั้น โดย ในอัลกอริทึ่มอื่นไม่มีอากาสเกิดเลย โดยจเป็นการ เท่าเดิมหรือ ลดลงเท่านั้น*

***จากสมตติฐานที่ 2*** *เพื่อทดสอบว่าการเกิด page fault ในกรณีที่เพิ่มจำนวน process แต่ไม่เพิ่ม page จะทำให้เกิด page fault ใน OPT น้อยกว่า อีก 2 แบบ*

***จากสมตติฐานที่ 3*** *เพื่อทดสอบว่า เมื่อใช้ค่า Random Reference String เมื่อสุ่มลำดับแล้ว ผลที่ออกมา จะมีค่าผลลัพธ์ ของ OPT ที่ได้ผลลัพธ์ มากที่สุด ในทุกๆ Process และ ทุกๆ Page เสมอ*

***จากสมตติฐานที่ 4*** *เพื่อทดสอบว่า เมื่อใช้ค่า Random Reference String เมื่อสุ่มลำดับแล้ว ผลที่ออกมาทั้ง 3 อัลกอริทึ่มนั้น ถึงแม้ว่าจะมีบาง อัลกอริทึ่มที่ได้ค่า มากกว่าหรือน้อยกว่า แต่ถ้ามองดูรวมๆแล้ว ความต่อเนื่องของกราฟ จะค่อนข้างคล้ายกันเสมอ*

***จากสมตติฐานที่ 5*** *เพื่อทดสอบว่า ไม่ว่าเราจะจัดเรียงแบบใดเรากะได้ค่าที่ออกจาก OPT เท่าเดิมเสมอ เนื่องจากเป็นการนำค่ามาจัดเรียงใหม่ทำให้ยังไงก็จะได้ค่าเดิมเสมอทุกๆการคำนวณ*