ตัวควบคุมและคลาสอรรถประโยชน์สำหรับการวิเคราะห์อัลกอริทึมเชิงทดลอง

นายวิริยะ นิลละออ



วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต
สาขาวิชาวิทยาศาสตร์คอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2552

# CONTROLLER AND UTILLITY CLASS FOR EXPERIMENTAL ALGORITHM ANALYSIS

Mr. Wiriya Nilla-or



A Thesis Submitted in Partial Fulfillment of the Requirements

for the Degree of Master of Science Program in Computer Science

Department of Computer Engineering

Faculty of Engineering

Chulalongkorn University

Academic Year 2009

| | |
|---|---|
| หัวข้อวิทยานิพนธ์ | ตัวควบคุมและคลาสอรรถประโยชน์สำหรับการวิเคราะห์อัลกอริทึมเชิงทดลอง |
| โดย | นายวิริยะ นิลละออ |
| สาขาวิชา | วิทยาศาสตร์คอมพิวเตอร์ |
| อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก | รองศาสตราจารย์ ดร. สมชาย ประสิทธิ์จูตระกูล |

---

คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย อนุมัติให้นับวิทยานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญามหาบัณฑิต

..... คณบดีคณะวิศวกรรมศาสตร์
(รองศาสตราจารย์ ดร.บุญสม เลิศหิรัญวงศ์)

คณะกรรมการสอบวิทยานิพนธ์

..... ประธานกรรมการ
(ผู้ช่วยศาสตราจารย์ ดร.วีระ เหมืองสิน)

..... อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก
(รองศาสตราจารย์ ดร.สมชาย ประสิทธิ์จูตระกูล)

..... กรรมการ
(ผู้ช่วยศาสตราจารย์ ดร.สุกรี สินธุภิญโญ)

..... กรรมการภายนอกมหาวิทยาลัย
(ผู้ช่วยศาสตราจารย์ ดร.วรเศรษฐ สุวรรณิก)

วิริยะ นิลละออ : ตัวควบคุมและคลาสอรรถประโยชน์สำหรับการวิเคราะห์อัลกอริทึมเชิงทดลอง. (CONTROLLER AND UTILLITY CLASS FOR EXPERIMENTAL ALGORITHM ANALYSIS) อ.ที่ปรึกษาวิทยานิพนธ์หลัก : รองศาสตราจารย์ ดร.สมชาย ประสิทธิ์จูตระกูล, 76 หน้า.

วิทยานิพนธ์ฉบับนี้นำเสนอ ตัวควบคุมและคลาสอรรถประโยชน์สำหรับการวิเคราะห์อัลกอริทึมเชิงทดลอง ที่เหมาะสำหรับการนำเสนอเนื้อหาทางด้านประสิทธิภาพเชิงเวลาของอัลกอริทึมที่สนใจ โดยไม่ต้องนำเสนอการวิเคราะห์ทางคณิตศาสตร์ ตัวควบคุมนี้อาศัยการแทรกคำสั่งเก็บข้อมูลที่ต้องการวิเคราะห์พฤติกรรมของโปรแกรมจาวาในรหัสต้นฉบับ ทำให้ทำงานได้เร็วเต็มที่ ส่งผลให้สามารถทำการทดลองกับปริมาณข้อมูลขนาดใหญ่ได้ นอกจากนี้ยังรองรับการกำหนดรูปแบบการทดลองกำกับเมท็อดที่สนใจ อันประกอบด้วย ลักษณะและปริมาณข้อมูลขาเข้า และรูปแบบการแสดงหรือบันทึกผลระหว่างการทำการทดลองตัวควบคุมการทดลองจะจัดการป้อนข้อมูลขาเข้า นับจำนวนครั้งที่คำสั่งทำงาน บันทึกผล และวนทำการทดลองตามรูปแบบที่กำหนดไว้อย่างอัตโนมัติ



ภาควิชา วิศวกรรมคอมพิวเตอร์ ลายมือชื่อนิสิต .....

สาขาวิชา วิทยาศาสตร์คอมพิวเตอร์ ลายมือชื่ออ.ที่ปรึกษาวิทยานิพนธ์หลัก .....

ปีการศึกษา 2552

## 507 14463 21 : MAJOR COMPUTER SCIENCE

KEYWORDS : ALGORITHM ANALYSIS / EXPERIMENTAL ANALYSIS

WIRIYA NILLA-OR : CONTROLLER AND UTILLITY CLASS FOR EXPERIMENTAL ALGORITHM ANALYSIS. THESIS ADVISOR : ASSOCIATE PROFESSOR SOMCHAI PRASITJUTRAKUL,Ph.D., 76 pp.

This paper presents a controller program for experimental analysis of algorithm session. It is designed to be used for presenting algorithm analysis topics without doing any mathematical analysis. It adds instrumentation instructions at the source code level in Java to observe interested algorithm behaviors. This allows the instrument the method with large size of data sets. In addition, It supports user's experiment configurations by using method annotation to specify input data size and characteristics, and presentation formats of experimental results. The system automatically setup input, invoke the instrument method, and report result as specified in the configuration.



Department : Computer Engineering Student's Signature .....

Field of Study : Computer Science Advisor's Signature .....

Academic Year : 2009

# กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้สำเร็จลุล่วงไปด้วยความช่วยเหลืออย่างดียิ่งของอาจารย์ที่ปรึกษาคือรองศาสตราจารย์ ดร.สมชาย ประสิทธิ์จูตระกูล ที่ให้คำปรึกษา ความรู้ ดูแลและให้ข้อเสนอแนะเกี่ยวกับงานวิจัยด้วยดีตลอด รวมทั้งช่วยเหลือ และสละเวลา เพื่อให้งานนี้เสร็จลุล่วงไปด้วยดี

ท้ายนี้ ผู้วิจัยขอกราบขอบพระคุณมารดา ซึ่งสนับสนุนทั้งด้านทุนทรัพย์ในการศึกษาและคอยให้กำลังใจแก่ผู้วิจัยเสมอมาจนกระทั่งสำเร็จการศึกษา

# สารบัญ

หน้า

# สารบัญตาราง

หน้า

# สารบัญภาพ

หน้า

# สารบัญรหัส

หน้า

# บทที่ 1
# บทนำ

## 1.1 ความเป็นมาและความสำคัญของปัญหา

การวิเคราะห์อัลกอริทึมมีจุดประสงค์เพื่อศึกษาประสิทธิภาพเชิงเวลาการทำงานของอัลกอริทึมว่า มีความสัมพันธ์อย่างไรกับขนาดของข้อมูลขาเข้า หรือกับลักษณะของข้อมูลขาเข้า [1] (นอกจากนี้ผู้วิเคราะห์อาจสนใจปริมาณหน่วยความจำที่ใช้ในการทำงานของอัลกอริทึม แต่ในงานวิจัยนี้จะนำเสนอเฉพาะประเด็นเรื่องเวลาการทำงานเท่านั้น) โดยทั่วไปการวิเคราะห์อาจเป็นแบบการวิเคราะห์กรณีช้าสุด หรือการวิเคราะห์กรณีเฉลี่ย การวิเคราะห์ทั้งสองอาศัยการเขียนความสัมพันธ์หรือฟังก์ชันทางคณิตศาสตร์ที่แทนจำนวนครั้ง (มากสุด หรือเฉลี่ย) ที่คำสั่งของอัลกอริทึมทำงาน โดยอาจอาศัยการนับเฉพาะคำสั่งตัวแทนบางคำสั่ง (ที่จำนวนครั้งของการทำงานมีความสัมพันธ์กับเวลาทำงาน) และอาศัยการวิเคราะห์เชิงเส้นกำกับ (asymptotic analysis) ให้ได้ผลที่สะท้อนอัตราการเติบโตของฟังก์ชัน ซึ่งเพียงพอกับการเปรียบเทียบประสิทธิภาพเชิงเวลาของอัลกอริทึมต่าง ๆ ได้ แต่การวิเคราะห์เชิงคณิตศาสตร์ผู้วิเคราะห์จะต้องลงไปในรายละเอียดซึ่งมีความซับซ้อน

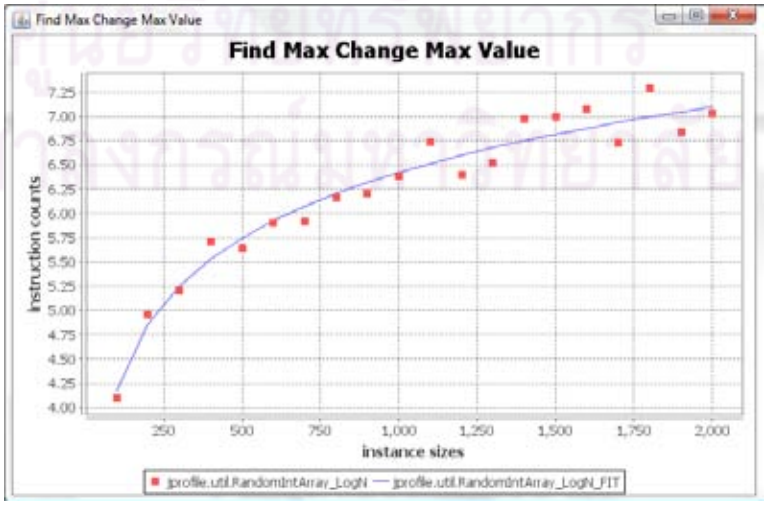
นอกจากการวิเคราะห์ที่ได้กล่าวข้างต้น ผู้วิเคราะห์อาจใช้วิธีการวิเคราะห์เชิงทดลอง [2] ซึ่งใช้วิธีเขียนโปรแกรมที่ทำตามอัลกอริทึมที่สนใจ ทดลองสั่งทำงานจริง แล้วเก็บข้อมูลที่สนใจระหว่างการทำงาน เช่น บันทึกเวลาการทำงานจริงกำกับข้อมูลที่ใช้ทดสอบการทำงาน หรือไม่ก็ใช้วิธีการนับจำนวนครั้งที่คำสั่งที่สนใจทำงาน ทำการทดลองเช่นนี้กับข้อมูลขาเข้าที่หลากหลาย เพื่อหาความสัมพันธ์ของเวลาการทำงานกับปริมาณหรือลักษณะของข้อมูลขาเข้าและได้เห็นพฤติกรรมระหว่างทำงานจริงของอัลกอริทึมที่สนใจด้วย

งานวิจัยนี้นำเสนอระบบช่วยการวิเคราะห์อัลกอริทึมเชิงทดลอง ที่เหมาะกับการใช้ประกอบการเรียนการสอนเนื้อหาที่เกี่ยวกับประสิทธิภาพเชิงเวลาของอัลกอริทึมที่สนใจ โดยไม่ต้องการลงในรายละเอียดของการวิเคราะห์ทางคณิตศาสตร์ ตัวระบบอาศัยกลวิธีการแทรกคำสั่งในรหัสต้นฉบับเพื่อเก็บข้อมูลและอาศัยการเขียน annotation [3] กำกับเมท็อดที่เขียนด้วยภาษาจาวาเพื่อบรรยายรูปแบบการทดลอง อันประกอบด้วย ลักษณะและปริมาณของข้อมูลขาเข้า พร้อมทั้งรูปแบบการแสดงหรือบันทึกผล เมื่อสั่งตัวควบคุมทำงาน ตัวควบคุมการจะจัดการป้อนข้อมูลขาเข้า นับจำนวนครั้งที่คำสั่งทำงาน บันทึกผล และวนทำการทดลองตามรูปแบบที่กำหนดไว้อย่างอัตโนมัติ

ตัวอย่างเช่น ผู้วิเคราะห์สนใจหาค่าเฉลี่ยของการเปลี่ยนค่ามากสุดระหว่างการหาค่ามากสุดในอาเรย์ การวิเคราะห์เชิงทดลองกระทำได้ด้วยการเขียนโปรแกรมตามอัลกอริทึม ดังรหัสที่ 1.1 เขียนกำหนดข้างหน้าคำสั่งที่สนใจนับ (บรรทัดที่ 14) ในที่นี้คือคำสั่ง `max = data[i]` และเขียนลักษณะของการทดลองกำกับที่หัวเมท็อดที่สนใจ (บรรทัดที่ 2 ถึง 8) จากตัวอย่างกำหนดให้ทำการทดลองด้วยข้อมูลขาเข้าเป็นอาเรย์ของจำนวนเต็มสุ่ม (บรรทัดที่ 4) มีขนาดเริ่มจาก 100 ข่องไปจนถึง 2,000 ช่อง เพิ่มทีละ 100 ช่อง ทำการทดลองซ้ำ ๆ 300 ครั้งสำหรับ แต่ละขนาดของข้อมูล (บรรทัดที่ 6) ให้นำผลของการนับจำนวนคำสั่งที่สนใจทั้งหมดไปแสดงผลออกเป็นกราฟ (บรรทัดที่ 5) รูปที่ 1.1 แสดงผลการทดลองที่ได้ เห็นได้ว่า มีแนวโน้มแบบ log สอดคล้องกับการวิเคราะห์ ทางคณิตศาสตร์ของ Knuth ว่า จำนวนการเปลี่ยนค่ามากสุดโดยเฉลี่ยมีค่าเท่ากับ $H_n - 1$ โดยที่ $H_n$ คือจำนวนฮาร์มอนิกที่ $n$ มีค่าประมาณ $\ln n$ เมื่อ $n$ มีค่ามาก [4]

```
01: public class FindMax {
02:  @Profile(name = "Find Max Change Max Value",
03:           inputs = {jprofile.util.RandomIntArray.class},
04:           outputs = {jprofile.output.ScatterPlot.class},
05:           from = 100, to = 2000, step=100, repeat=300,
06:           curveFitting= true(
07:  public int max(int[] data) {
08:    int max = data[0];
09:    for (int i = 1; i < data.length; i++) {
10:      if (max < data[i]) {
11:        //@Profiler.counts[0]++;
12:        max = data[i];
13:      }
14:    }
15:    return max;
16:  }
17:}
```

รหัสที่ 1.1 การทดลองหาค่าเฉลี่ยการเปลี่ยนค่ามากสุด




รูปที่ 1.1 ผลการทดลองหาค่าเฉลี่ยค่ามากสุด แนวโน้มแบบ log n

## 1.2 วัตถุประสงค์ของการวิจัย

ออกแบบและพัฒนาตัวควบคุมและคลาสอรรถประโยชน์สำหรับการวิเคราะห์อัลกอริทึมเชิงทดลอง ที่อาศัยการแทรกคำสั่งเก็บข้อมูลในโปรแกรม

## 1.3 ขอบเขตของการวิจัย

1. พัฒนาระบบด้วยภาษาจาวา
2. ระบบช่วยวิเคราะห์อัลกอริทึมที่เขียนเป็นโปรแกรมภาษาจาวา
3. ระบบมีคลาสผลิตข้อมูลประเภทจำนวน และอาเรย์ของจำนวน
4. ระบบมีคลาสแสดงผลการทดลองเป็น กราฟการกระจาย (scatter plot), เป็นข้อความทาง stdout และเป็นแฟ้มข้อความ
5. ระบบอาศัย annotation ของจาวาในการระบุลักษณะการทดลอง

## 1.4 ประโยชน์ที่คาดว่าจะได้รับ

ได้ระบบเสริมการวิเคราะห์อัลกอริทึมเชิงทดลองที่สามารถใช้เสริมการทำความเข้าใจพฤติกรรมเชิงประสิทธิภาพของอัลกอริทึมที่สนใจ สามารถใช้ประกอบการเรียนการสอนวิชาต่าง ๆ ที่เกี่ยวกับการวิเคราะห์อัลกอริทึมได้

## 1.5 วิธีดำเนินการวิจัย

1. ศึกษาการวิเคราะห์อัลกอริทึมเชิงทดลอง
2. ศึกษาคลาสจาวาเพื่ออำนวยความสะดวกในการพัฒนาระบบ
3. ออกแบบตัวควบคุมและคลาสอรรถประโยชน์ของระบบ
4. พัฒนาตัวควบคุมและคลาสอรรถประโยชน์ที่ออกแบบไว้
5. เขียนตัวอย่างการวิเคราะห์อัลกอริทึมที่น่าสนใจเป็นกรณีศึกษาและทดสอบระบบ
6. สรุปผลการวิจัยและจัดทำวิทยานิพนธ์เป็นรูปเล่ม

# บทที่ 2
# เอกสารและงานวิจัยที่เกี่ยวข้อง

## 2.1 แนวคิดและทฤษฎี

### 2.1.1 การวิเคราะห์อัลกอริทึม

การวิเคราะห์อัลกอริทึม คือ การประมาณทรัพยากร เช่น เวลา, หน่วยความจำ ที่อัลกอริทึมใช้ในการทำงาน ในงานวิจัยนี้จะเน้นเฉพาะเรื่องเวลาว่า มีความสัมพันธ์อย่างไรกับขนาดของข้อมูลขาเข้า หรือลักษณะของข้อมูลขาเข้า อาศัยการนับจำนวนครั้งที่คำสั่งทำงานเพราะเป็นตัวสะท้อนเวลาการทำงานของอัลกอริทึม แบ่งได้ 2 วิธีคือ

#### 2.1.1.1 การนับทุกคำสั่ง

วิธีนี้นับคำสั่งทั้งหมดในอัลกอริทึมที่ทำงาน จากรหัสที่ 2.1 แสดงให้เห็นว่าการหาข้อมูลใน อาเรย์ กรณีช้าสุดทำงานทั้งสิ้น $3n + 1$ ครั้ง ($n$ คือขนาดของอาเรย์)

```
public class Experiment1 {
  static boolean arraySearch(int[] d , int v) {
    int max = d[0];                              <----  1
    for (int i = 1; i < d.length; i++) {         <----  (1) + (n) + (n-1)
      if (v == d[i]) {                           <----  n-1
        return true;                             <----  0  (กรณีช้าสุด คือหาไม่พบ)
      }
    }
    return false;                          <----  1
  }
}                                          รวมทั้งสิ้น   3n + 1 ครั้ง
```

รหัสที่ 2.1 ตัวอย่างการวิเคราะห์โดยการนับคำสั่งทั้งหมด กรณีช้าสุด

#### 2.1.1.2 การนับเฉพาะคำสั่งตัวแทน

การนับเฉพาะคำสั่งตัวแทน เป็นการนับคำสั่งที่เป็นตัวแทนของการทำงาน โดยเวลาการทำงานรวมของอัลกอริทึม จะมีแนวโน้มการเติบโตของเวลาการทำงานเช่นเดียวกับเมื่อนับเฉพาะคำสั่งตัวแทน ทั้งนี้เพื่อลดความซับซ้อนของการนับ รหัสที่ 2.2 แสดงผลการนับคำสั่งตัวแทน ซึ่งคือคำสั่งเปรียบเทียบในกรณีช้าสุด เกิดการเปรียบเทียบข้อมูลทุกตัวในอาเรย์เป็นจำนวน $n - 1$ ครั้ง ($n$ คือขนาดของอาเรย์) เมื่อเทียบกับการนับทุกคำสั่งที่ได้ $3n + 1$ ครั้ง ถือได้ว่า มีแนวโน้มการเติบโตของฟังก์ชันเวลาการทำงานเป็นเชิงเส้นเหมือนกัน กล่าวว่ามีเวลาการทำงานเชิงเส้นกำกับในกรณีช้าสุดเป็น $\Theta(n)$ ทั้งการนับทุกคำสั่ง กับการนับเฉพาะคำสั่งตัวแทน

```
public class Experiment1 {
  static boolean arraySearch(int[] d , int v) {
    int max = d[[0;
    for (int i = 1; i < d.length; i++) {
      if (v == d[i]) {          ← นับเฉพาะคำสั่ง v == d[i] ได้ n-1 ครั้ง
        return true;
      }
    }
    return false;
  }
}
```

รหัสที่ 2.2 ตัวอย่างการวิเคราะห์โดยการนับเฉพาะคำสั่งตัวแทน กรณีช้าสุด

การนับคำสั่งทั้ง 2 วิธีนั้นอาจจะนับโดยวิธีการทางคณิตศาสตร์ หรือการนับโดยการทดลอง โดยทั่วไปการวิเคราะห์อาจเป็น การวิเคราะห์กรณีช้าสุด คือ เวลาที่อัลกอริทึมใช้เวลามากที่สุด หรือ การวิเคราะห์กรณีเฉลี่ย คือ เวลาเฉลี่ยที่คาดว่าอัลกอริทึมใช้ในการทำงาน

### 2.1.2 โพรไฟล์เลอร์

การวิเคราะห์พฤติกรรมของโปรแกรมขณะทำงานอาศัยเครื่องมือที่เรียกว่า โพรไฟล์เลอร์ (Profiler) การเก็บข้อมูลระหว่างการทำงานของโพรไฟล์เลอร์แบ่งได้ 3 วิธีคือ

1. การเก็บข้อมูลตามเหตุการณ์ที่สนใจ (event-based) เช่น เมื่อเมท็อดที่สนใจทำงานหรือเลิกทำงาน เมื่อจบการทำงานทีละคำสั่ง เป็นต้น วิธีนี้มักทำให้โปรแกรมทำงานช้ามาก
2. การชักตัวอย่าง (sampling) เป็นการเก็บข้อมูลเป็นระยะ ๆ จาก stack trace ของโปรแกรม ถึงแม้ว่าโปรแกรมจะถูกขัดจังหวะเพื่อให้เก็บข้อมูล แต่โปรแกรมยังสามารถทำงานได้เร็วเต็มที่อย่างไรก็ตามข้อมูลที่เก็บได้เป็นค่าประมาณเชิงสถิติเท่านั้น
3. การแทรกรหัสคำสั่งเก็บข้อมูลในโปรแกรม (code instrumentation) โดยผู้เขียนโปรแกรมเป็นผู้แทรกเอง หรือให้ระบบแทรกขณะทำงาน

โพรไฟล์เลอร์ที่เก็บข้อมูลระหว่างทำงานของโปรแกรมจาวานั้น มักอาศัยกลไกที่ระบบจาวามีให้คือ JVMPI (JVM profiling Interface) [5] และ JVMTI (JVM Tool Interface) [6] อาทิเช่น HPROF [7], Visual VM [8] ,Netbeans Profiler [9] เป็นต้น แต่การพัฒนาโพรไฟล์เลอร์ในลักษณะนี้จะเป็นแบบที่ขึ้นกับระบบ [10] ซึ่งขัดกับจุดมุ่งหมายของจาวา ที่ควรพัฒนาครั้งเดียวแต่ทำงานได้ในหลายระบบ ในขณะที่ระบบ JP [10] และ ByCounter [11] อาศัยการแทรกรหัสคำสั่งเก็บข้อมูลในรหัสไบต์ (bytecode) ของคลาสด้วยจาวาล้วน ๆ จึงสามารถทำงานได้บนระบบจาวาไม่ขึ้นกับระบบของเครื่องที่ทำงาน นอกจากนี้คลาสที่ถูกแทรกรหัสคำสั่งยังสามารถทำงานได้เร็วเต็มที่

## 2.2 เอกสารและงานวิจัยที่เกี่ยวข้อง

### 2.2.1 JP

JP [10] เป็นโพรไฟล์เลอร์ที่วัดประสิทธิภาพการทำงานของโปรแกรมจาวาจากจำนวนที่รหัสไบต์ทำงาน การนับรหัสไบต์อาศัยการแปลงรหัสต้นฉบับ เป็นรหัสต้นฉบับใหม่ที่มีคำสั่งนับแทรกอยู่ รหัสที่ 2.3 แสดงตัวอย่างการแปลงรหัสต้นฉบับโดยการแทรกคำสั่งไปที่จุดเริ่มต้นของแต่ละกลุ่มคำสั่ง เช่น กลุ่ม `if(i <= 10)` มีรหัสไบต์ทำงานอยู่ภายใน 7 คำสั่ง จึงแทรกคำสั่ง `ic.instr += 7;` เพื่อนับจำนวนคำสั่งเพิ่มอีก 7 (สำหรับคลาส IC ที่เพิ่มเข้ามาด้วยนั้น มีไว้สำหรับเก็บข้อมูลลำดับการเรียกใช้เมท็อด)

```
void f() {
                          void f() {
                            f(RC.getRC());
                          }
                          void f(IC ic) {
                            ic.instr += 2;
  int i = 1;                int i = 1;
  while (true) {            while (true) {
                              ic.instr += 2;
    if (i <= 10) {            if (i <= 10) {
                                ic.instr += 7;
      h();                      h();
      g(i);                     g(i);
      ++i;                      ++i;
    } else {                  } else {
                                ic.instr += 1;
      return;                   return;
    }                         }
  }                         }
}                         }
```

```
aload_0
invokevirtual #2
aload_0
iload_1
invokevirtual #3
iinc 1
goto 2
```

รหัสที่ 2.3 ตัวอย่างการแปลงรหัสต้นฉบับของ JP

### 2.2.2 ByCounter

ByCounter [11] เป็นโพรไฟล์เลอร์ที่วัดประสิทธิภาพการทำงานของโปรแกรมจาวาจากจำนวนรหัสไบต์ที่ถูกทำงานเมื่อมีการทำงานจริงเช่นเดียวกับ JP แต่วิธีการแทรกรหัสไบต์ที่ทำหน้าที่นับคำสั่งนั้นกระทำที่รหัสไบต์แทนที่จะกระทำที่รหัสต้นฉบับของโปรแกรม

### 2.2.3 HPROF Heap/CPU Profiling Tool in J2SE 5.0

HPROF [7] เป็นโพรไฟล์เลอร์ที่ระบบจาวามีไว้ช่วยวัดการใช้ CPU และ heap ผ่านคำสั่ง command line โดยอาศัย JVMTI ที่เป็นชุดคำสั่งขอระบบแบบ native code เพื่อใช้เก็บ

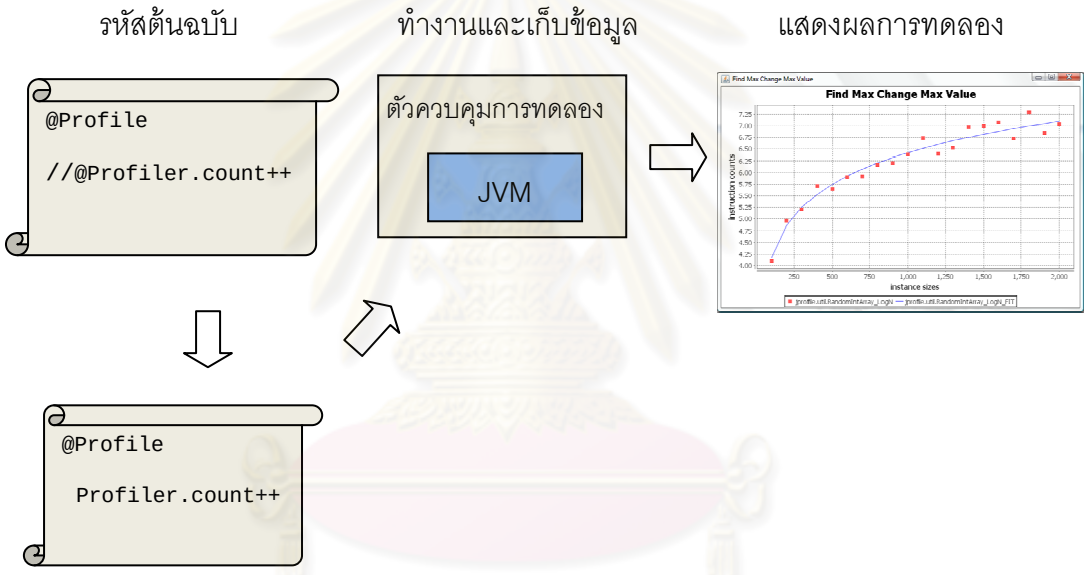
ข้อมูลการทำงานของ JVM เช่น การใช้ CPU, การใช้ heap เป็นต้น โดยโพรไฟล์เลอร์จะลงทะเบียนเหตุการณ์ที่สนใจไว้ที่ JVM เมื่อเหตุการณ์นั้นเกิดขึ้น JVM จะแจ้งกลับมาที่โพรไฟล์เลอร์ จากนั้นสามารถเรียกขอข้อมูลจาก JVM ผ่านฟังก์ชันที่มีให้ เนื่องจากจะต้องมีการร้องขอข้อมูลจาก JVM ดังนั้น สามารถใช้งาน HPROF ได้เฉพาะกับ JVM ที่รองรับ JVMTI เท่านั้น

# บทที่ 3

# การใช้งานตัวควบคุมการทดลองและคลาสอรรถประโยชน์

บทนี้จะกล่าวถึงการใช้งานตัวควบคุมการทดลองและคลาสอรรถประโยชน์ เพื่อใช้ในการวิเคราะห์อัลกอริทึมเชิงทดลอง อธิบายถึงภาพรวมการทำงานของตัวตวบคุมการทดลอง การใช้งานตัวควบคุม การกำหนดรูปแบบการทดลอง และการกำหนดรูปแบบการแสดงผลการทดลอง โดยแสดงผลการทดลองในรูปกราฟของปริมาณข้อมูลขาเข้ากับจำนวนการทำงานของคำสั่งตัวแทน เพื่อให้ผู้ใช้เห็นอัตราเติบโตได้ชัดเจน และเข้าใจการใช้งานตัวควบคุมการทดลอง

## 3.1 ภาพรวมการทำงานของตัวควบคุมการทดลอง




รูปที่ 3.1 ภาพรวมการทำงานของตัวควบคุมการทดลอง

การทำงานของตัวควบคุมการทดลองซึ่งแสดงในรูปที่ 3.1 ผู้ใช้เตรียมรหัสต้นฉบับที่กำกับ annotation ชื่อ @Profile ที่เมท็อดหรือคลาส และเขียน //@Profiler.count++ ไว้หน้าคำสั่งที่ต้องการนับจำนวนครั้งการทำงาน เมื่อเริ่มการทดลองตัวควบคุมจะนำรหัสต้นฉบับดังกล่าวไปทำการเปลี่ยน //@Profiler.count++ ให้เป็นคำสั่ง Profiler.count++ จากนั้นแปลรหัสต้นฉบับที่แก้ไขแล้วเป็นรหัสไบต์ใหม่ และย้ายไปยังตำแหน่งที่เหมาะสมเพื่อเตรียมให้ JVM นำเข้าหน่วยความจำได้ จากนั้นตัวควบคุมจะสั่งให้ตัวโปรแกรมทำงานและแสดงผลการทดลอง ตามที่กำหนดไว้ใน @Profile

## 3.2 ภาพรวมการใช้งานตัวควบคุมการทดลอง

การใช้งานตัวควบคุมการวิเคราะห์มี 3 ขั้นตอนได้แก่

### 3.2.1 การเตรียมระบบก่อนทำการวิเคราะห์เชิงทดลอง

เนื่องจากการวิเคราะห์เชิงทดลองนี้ ตัวควบคุมการทดลองทำงานโดยย้ายรหัสไบต์ที่ได้จากการคอมไพล์รหัสต้นฉบับที่แก้ไข //@Profiler.count++ ให้เป็นคำสั่ง Profiler.count++ ซึ่งสภาพแวดล้อมของระบบของผู้ใช้แตกต่างกัน จึงให้ผู้ใช้เป็นผู้กำหนดตำแหน่งของรหัสต้นฉบับและตำแหน่งของรหัสไบต์ใหม่ในไฟล์ที่ชื่อว่า profile.properties ซึ่งมีข้อมูล 2 ค่า คือ profile.source.dir ตำแหน่งของรหัสต้นฉบับ และ profile.class.dir ตำแหน่งของรหัสไบต์ใหม่ ดังแสดงตัวอย่างในรหัสที่ 3.1

```
profile.source.dir = src
profile.class.dir = build/classes
```

รหัสที่ 3.1 ตัวอย่างไฟล์ profile.properties

### 3.2.2 การเขียนรหัสต้นฉบับ

ผู้ใช้เขียนรหัสต้นฉบับตามอัลกอริทึมที่จะวิเคราะห์ จากนั้นกำกับ @Profile อธิบายลักษณะข้อมูลขาเข้า ลักษณะการทดลอง และลักษณะการแสดงผลการทดลอง ซึ่งจะกล่าวรายละเอียดการกำกับ @Profile ในหัวข้อที่ 3.3 และเขียน //@Profiler.count++ ไว้หน้าคำสั่งที่ต้องการนับ การที่ให้ผู้ใช้แทรกข้อความ //@Profiler.count++ ลงในรหัสต้นฉบับแล้วให้ตัวควบคุมเปลี่ยนเป็นคำสั่ง Profiler.count++ แทนที่จะให้ผู้ใช้แทรกคำสั่ง Profiler.count++ โดยตรง ทำให้ผู้ใช้สามารถใช้งานรหัสต้นฉบับนั้นได้โดยไม่ต้องใช้ตัวควบคุมการทดลอง

### 3.2.3 การสั่งให้ตัวควบคุมการทดลองทำงาน

ผู้ใช้สั่งให้ตัวควบคุมเริ่มทำงานได้โดยการเรียกใช้เมท็อด begin ของ คลาส Profiler โดยส่งพารามิเตอร์เป็นชื่อคลาสที่ต้องการวิเคราะห์ ดังตัวอย่างในรหัสที่ 3.2 สาเหตุที่กำหนดรับพารามิเตอร์เป็นข้อมูลชนิดสายอักขระ แทนการใช้คุณลักษณะ `class` ของคลาสที่ต้องการวิเคราะห์ เช่น `experimental.FindMax.class` เนื่องจาก JVM จะนำเข้ารหัสไบต์ของคลาสเข้าไปในหน่วยความจำ เมื่อมีการเรียกคลาสและไม่สามารถนำเข้ารหัสไบต์นั้นได้อีก ทำให้กระบวนการเปลี่ยนข้อความ //@Profiler.count++ เป็น Profiler.count++ แล้วแปลงเป็นรหัสไบต์ใหม่ไม่เกิดผล การใช้ชื่อคลาส `experimental.FindMax` เป็นข้อมูลชนิดสายอักขระ เป็นการส่ง

ข้อมูลคลาสที่ต้องการวิเคราะห์ให้กับตัวควบคุมการทดลอง โดยรหัสไบต์ของคลาสนั้นยังไม่ถูกนำเข้าหน่วยความจำของ JVM

```
new Profiler().begin("experimental.FindMax");
```

รหัสที่ 3.2 ตัวอย่างการสั่งให้ตัวควบคุมทำงาน

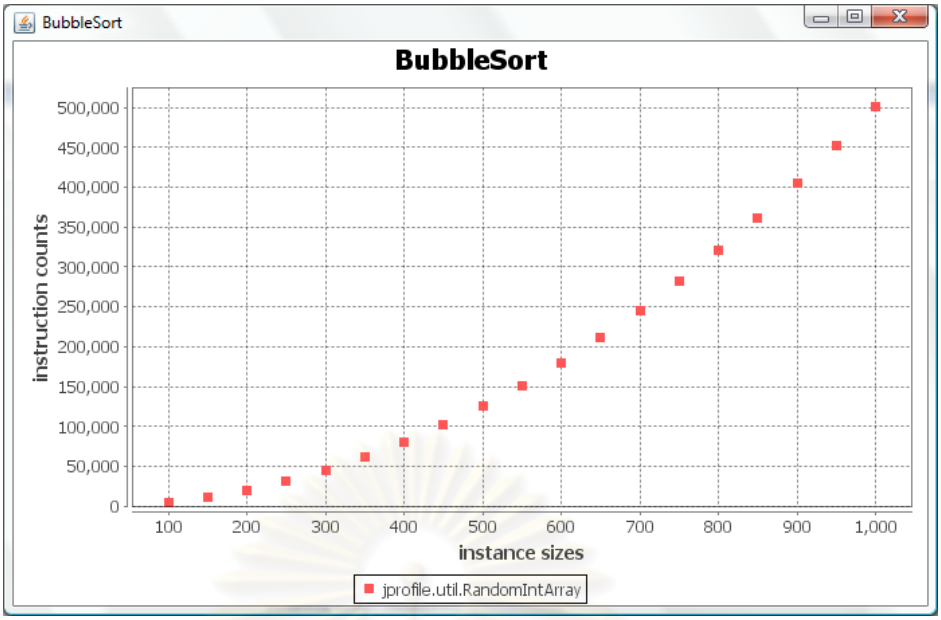
## 3.3 การกำกับ @Profile เพื่อกำหนดรูปแบบการทดลอง

ผู้ใช้สามารถกำหนด รูปแบบข้อมูลขาเข้า และ รูปแบบการทดลองได้โดยการเขียน @Profile กำกับไว้ที่เมท็อดที่สนใจ โดยรายละเอียดของการเขียน @Profile มีดังต่อไปนี้

### 3.3.1 การกำหนดรูปแบบการทดลองพื้นฐาน

ในการวิเคราะห์อัลกอริทึมการเรียงข้อมูลแบบบับเบิ้ล ที่สนใจว่าจำนวนครั้งของการเปรียบเทียบข้อมูลมีความสัมพันธ์กับจำนวนข้อมูลทั้งหมดอย่างไร โดยใช้ตัวควบคุมการวิเคราะห์เชิงทดลองแล้วแสดงผลการทดลองตามรูปที่ 3.2 ผู้ใช้สามารถทำได้โดย เขียนรหัสต้นฉบับ กำกับ @Profile และเขียน //@Profiler.count++ หน้าคำสั่งเปรียบเทียบข้อมูล ตามที่แสดงในรหัสที่ 3.3 ในส่วนของ @Profile ประกอบด้วย

- name : ระบุชื่อการทดลอง
- inputs : ระบุชื่อคลาสหน่วยผลิตข้อมูลขาเข้าที่จะใช้ในการทดลอง ในที่นี้ใช้ RandomIntArray ซึ่งผลิตข้อมูลอาเรย์แบบ int ที่มีค่าสุ่ม
- outputs : ระบุชื่อหน่วยแสดงผลข้อมูล ในที่นี้ใช้ ScatterPlot ซึ่งแสดงผลการทดลองด้วยกราฟ
- from, to, step : ระบุปริมาณข้อมูลที่ใช้ในการทดลอง ในที่นี้ให้เริ่มตั้งแต่อาเรย์ขนาด 100 ช่องทดลองซ้ำโดยเพิ่มครั้งละ 50 ช่อง จนกว่าอาเรย์จะมีขนาดเกิน 1,000 ช่อง
- repeat : ระบุจำนวนครั้งในการทดลองซ้ำ สำหรับแต่ละขนาดของข้อมูลขาเข้าตามที่กำหนดไว้ โดยในแต่ละรอบจะขอข้อมูลขาเข้าชุดใหม่จากหน่วยผลิตข้อมูลขาเข้า เพื่อให้ได้ผลการทดลองที่แม่นยำขึ้น




รูปที่ 3.2 จำนวนการเปรียบเทียบข้อมูลในการเรียงข้อมูลแบบบับเบิ้ล

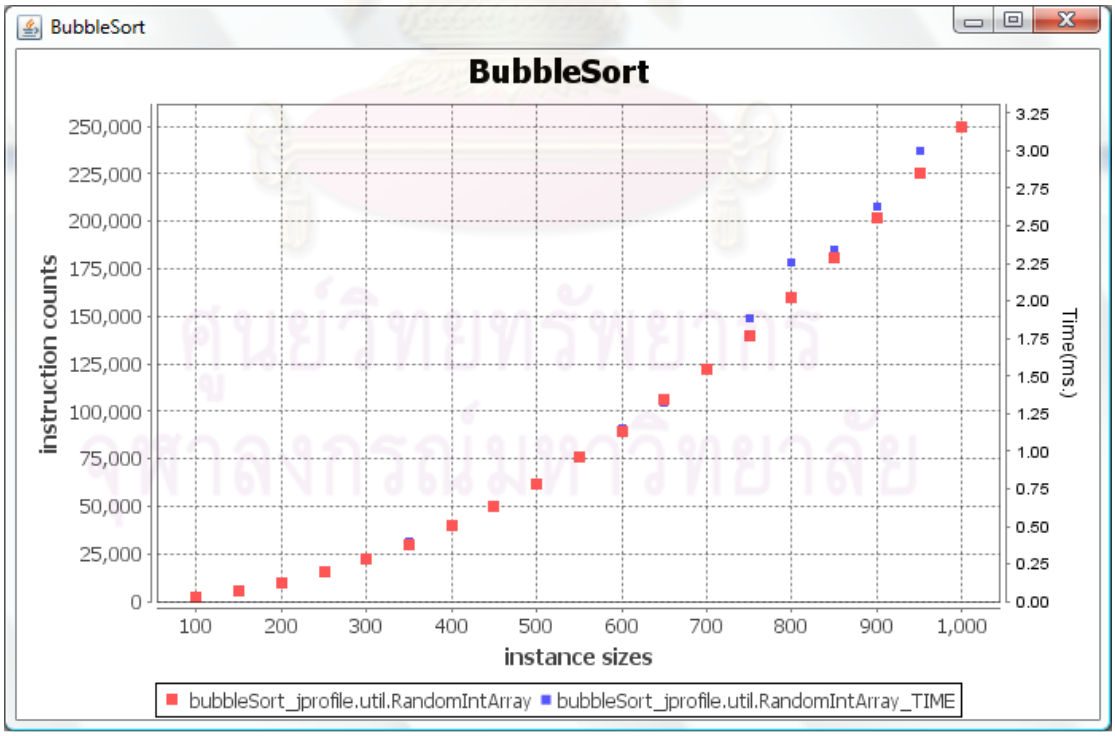
```
import jprofile.*;
import jprofile.util.*;

public class Sort {
    @Profile(name = "BubbleSort",
             inputs = { jprofile.util.RandomIntArray.class},
             outputs = {jprofile.output.ScatterPlot.class},
             from = 100, to = 1000, step = 50, repeat = 100)
    public int[] bubbleSort(int[] data){
        for(int i = 0;i<data.length;i++){
            for(int j = i ;j < data.length ; j++){
                //@Profiler.count++;
                if(data[i] > data[j]){
                    int temp = data[i];
                    data[i] = data[j];
                    data[j] = temp;
                }
            }
        }
        return data;
    }
}
```

รหัสที่ 3.3 การกำหนดรูปแบบการทดลองการเรียงข้อมูลแบบบับเบิ้ล

### 3.3.2 การแสดงเวลาการทำงานของโปรแกรมโดยเฉลี่ย

ในการศึกษาพฤติกรรมการทำงานของบางอัลกอริทึม นอกจากต้องการทราบจำนวนครั้งที่คำสั่งตัวแทนทำงานโดยเฉลี่ย ยังต้องการทราบเวลาทำงานเฉลี่ยของโปรแกรมด้วยว่าสัมพันธ์กับขนาดข้อมูลขาเข้าอย่างไร ดังแสดงในรูปที่ 3.3 โดยผู้ใช้เพิ่มการกำกับ @Profile ซึ่งแสดงในรหัสที่ 3.4 ประกอบด้วย

- displayTime : ระบุความต้องการให้ระบบแสดงข้อมูลเวลาการทำงานหรือไม่
- timeTruncate : ระบุจำนวนร้อยละของข้อมูลเวลาการทำงาน ที่จะถูกตัดออก ไม่นำมาคิดค่าเฉลี่ย โดยจะตัดข้อมูลที่มีค่ามากและค่าน้อยจากเวลาการทำงานของการทดลองซ้ำทั้งหมด ที่กำกับด้วย repeat ในแต่ละขนาดของข้อมูลขาเข้า เพราะขณะทดลองในแต่ละรอบ JVM อาจมีการทำงานอื่น เช่น จองหน่วยความจำ หรือ คืนหน่วยความจำนอกเหนือจากทำงานตามโปรแกรมที่ต้องการทดลอง ซึ่งจะทำให้การทำงานรอบนั้นใช้เวลามากกว่าปกติ ถ้านำเวลาดังกล่าวมาคิดค่าเฉลี่ยด้วยจะทำให้ได้ข้อมูลที่ผิดพลาด ดังนั้นจึงตัดข้อมูลดังกล่าวออก



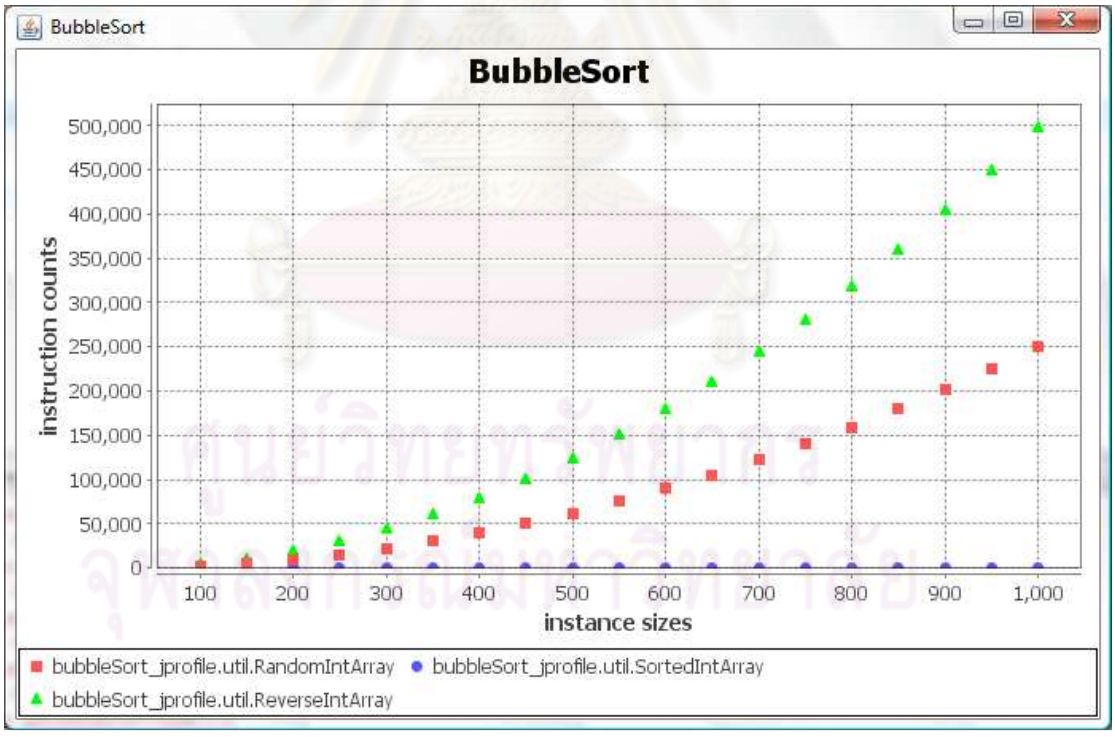

รูปที่ 3.3 จำนวนการเปรียบเทียบข้อมูลในการเรียงข้อมูลแบบบับเบิ้ล พร้อมแสดงเวลาการทำงาน

```
...
public class Sort {
    @Profile(name = "BubbleSort",
    inputs = {jprofile.util.RandomIntArray.class},
    outputs = {jprofile.output.ScatterPlot.class},
    from = 100, to = 1000, step = 50, repeat = 100,
    displayTime = true, timeTruncate = 5)
    public int[] bubbleSort(int[] data){
        ...
    }
}
```

รหัสที่ 3.4 การกำหนดรูปแบบการทดลองให้แสดงเวลาการทำงานโดยเฉลี่ย

### 3.3.3 การใช้หน่วยผลิตข้อมูลขาเข้าหลายตัว

ในการวิเคราะห์อัลกอริทึมการเรียงข้อมูลแบบบับเบิ้ล ที่ต้องการเปรียบเทียบจำนวนครั้งการสลับข้อมูลโดยใช้ข้อมูลขาเข้าแบบสุ่ม แบบเรียงจากน้อยไปหามาก และแบบเรียงจากมากไปหาน้อย ดังแสดงในรูปที่ 3.4 ผู้ใช้สามารถกำกับหน่วยผลิตข้อมูลขาเข้าด้วยค่า inputs ใน @Profile และเขียน //@Profiler.count++ หน้าคำสั่งสลับข้อมูล ซึ่งแสดงในรหัสที่ 3.5



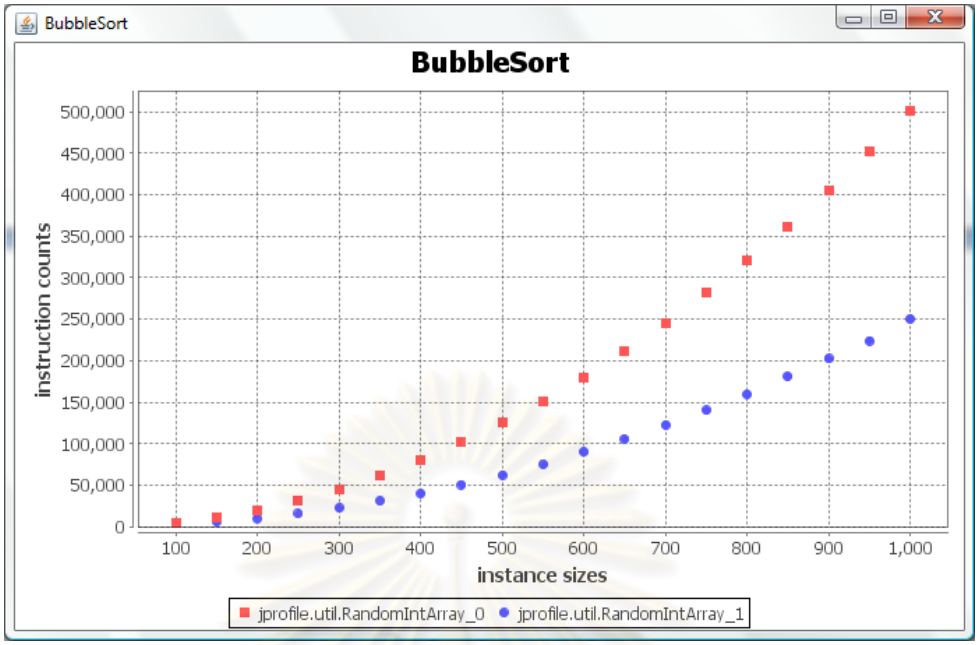

รูปที่ 3.4 จำนวนการสลับข้อมูลในการเรียงข้อมูลแบบบับเบิ้ล เปรียบเทียบการใช้ข้อมูลขาเข้าจากหน่วยผลิตข้อมูลขาเข้าแบบสุ่ม เรียงจากน้อยไปหามาก และจากมากไปหาน้อย

```
...
public class Sort {
    @Profile(name = "BubbleSort",
             inputs = {jprofile.util.RandomIntArray.class,
                       jprofile.util.SortedIntArray.class,
                       jprofile.util.ReverseIntArray.class},
             outputs = {jprofile.output.ScatterPlot.class},
             from = 100, to = 1000, step = 50, repeat = 100)
    public int[] bubbleSort(int[] data){
        for(int i = 0;i<data.length;i++){
            for(int j = i ;j < data.length ; j++){
                if(data[i] > data[j]){
                    //@Profiler.count++;
                    int temp = data[i];
                    data[i] = data[j];
                    data[j] = temp;
                }
            }
        }
        return data;
    }
}
```

รหัสที่ 3.5 การกำหนดรูปแบบการทดลองให้ใช้หน่วยผลิตข้อมูลขาเข้าหลายตัว

### 3.3.4 การใช้ตัวนับหลายตัว

ในการวิเคราะห์อัลกอริทึมการเรียงข้อมูลแบบบับเบิ้ล ที่สนใจจำนวนครั้งการเปรียบเทียบข้อมูลและจำนวนครั้งการสลับข้อมูล ดังรูปที่ 3.5 เห็นได้ว่าจะใช้ตัวนับสองตัวเพื่อเก็บจำนวนครั้งการเปรียบเทียบข้อมูล และจำนวนครั้งการสลับข้อมูล เมื่อต้องการใช้ตัวนับหลายตัว ตัวนับจะถูกเขียนอยู่ในรูปของอาเรย์ คือ ผู้ใช้สามารถเขียน //@Profiler.counts[n] ไว้หน้าคำสั่งที่สนใจ โดยที่ n คือหมายเลขของตัวนับ โดยที่ n มีค่าเริ่มจาก 0 และต้องเพิ่มการกำกับค่า countNo ใน @Profile เพื่อระบุจำนวนตัวนับที่ใช้ในการทดลอง ดังแสดงใน รหัสที่ 3.6 เมื่อมีการใช้ตัวนับหลายตัว ในคำอธิบายสัญลักษณ์จะเพิ่มหมายเลขตัวนับเข้าไปด้วย เพื่ออธิบายว่าข้อมูลชุดนั้นเป็นข้อมูลจากตัวนับตัวใด




รูปที่ 3.5 จำนวนการเปรียบเทียบและการสลับข้อมูลในการเรียงข้อมูลแบบบับเบิ้ล

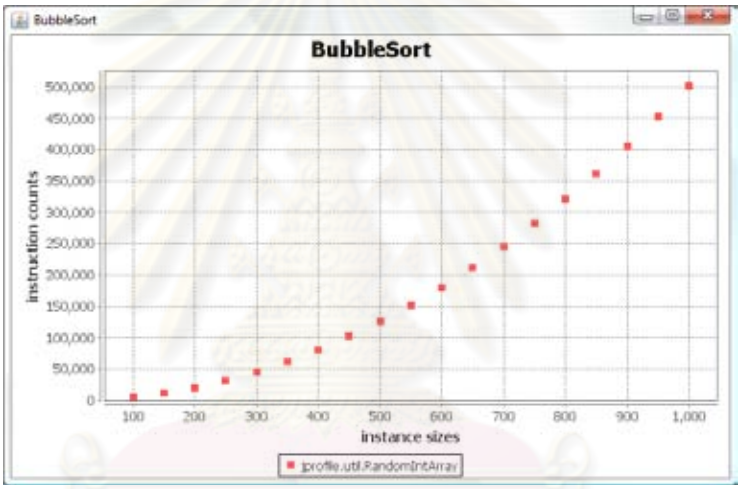
```
...
public class Sort {
    @Profile(name = "BubbleSort",
             inputs = {jprofile.util.RandomIntArray.class},
             outputs = {jprofile.output.ScatterPlot.class},
             from = 100, to = 1000, step = 50, repeat = 100,
             countNo = 2)
    public int[] bubbleSort(int[] data) {
        for (int i = 0; i < data.length; i++) {
            for (int j = i; j < data.length; j++) {
                //@Profiler.counts[0]++;
                if (data[i] > data[j]) {
                    //@Profiler.counts[1]++;
                    int temp = data[i];
                    data[i] = data[j];
                    data[j] = temp;
                }
            }
        }
        return data;
    }
}
```

รหัสที่ 3.6 การกำหนดรูปแบบการทดลองโดยใช้ตัวนับหลายตัว

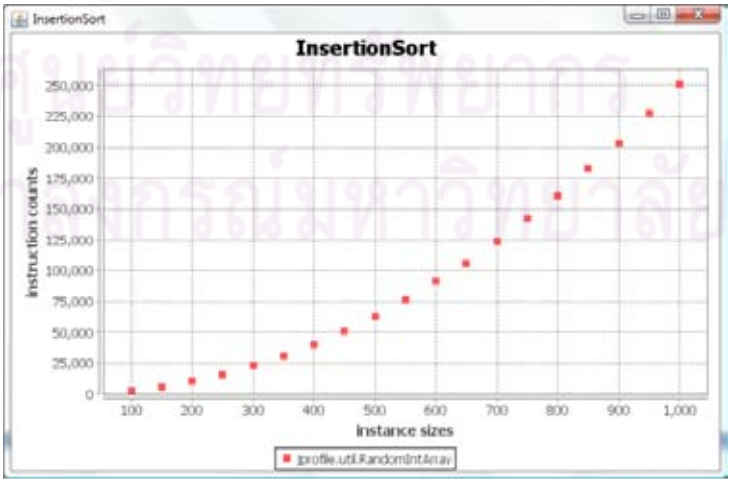
### 3.3.5 การวิเคราะห์เชิงทดลองพร้อมกันหลาย ๆ เมท็อด

ในการวิเคราะห์เชิงทดลองที่ต้องการเปรียบเทียบการทำงานของหลาย ๆ เมท็อด เช่น เปรียบเทียบจำนวนการเปรียบเทียบข้อมูลระหว่างการเรียงข้อมูลแบบบับเบิ้ล กับการเรียงข้อมูลแบบแทรก ดังรูปที่ 3.6 และ รูปที่ 3.7 ผู้ใช้สามารถเพิ่มการกำกับ @Profile ระดับคลาสโดยสามารถกำหนดค่าที่ทุกเมท็อดใช้ร่วมกันที่ระดับคลาสได้ และต้องกำกับ @Profile ระดับเมท็อดหน้าเมท็อดที่ต้องการทำการทดลองพร้อมทั้งกำหนดค่าที่ใช้เฉพาะเมท็อดนั้น ๆ ดังแสดงในรหัสที่ 3.7 โดยลำดับการใช้ค่าที่กำหนดไว้ใน @Profile เริ่มจากระดับเมท็อด ไปที่ระดับคลาส และถ้าไม่กำหนดในระดับคลาสจะใช้ค่าโดยปริยาย

- seed : กำหนดค่าเริ่มต้นที่ใช้ผลิตเลขสุ่มในการสร้างข้อมูลขาเข้า เพื่อให้ทุกเมท็อดได้รับชุดข้อมูลขาเข้าเหมือนกัน




รูปที่ 3.6 จำนวนการเปรียบเทียบในการเรียงข้อมูลแบบบับเบิ้ล




รูปที่ 3.7 จำนวนการเปรียบเทียบในการเรียงข้อมูลแบบแทรก

```
...
@Profile(seed = 1000L,
         inputs = {jprofile.util.RandomIntArray.class},
         from = 100, to = 1000, step = 50, repeat = 100)
public class Sort {
    @Profile(name = "BubbleSort",
             outputs = {jprofile.output.ScatterPlot.class})
    public int[] bubbleSort(int[] data) {
        for (int i = 0; i < data.length; i++) {
            for (int j = i; j < data.length; j++) {
                //@Profiler.count++;
                if (data[i] > data[j]) {
                    int temp = data[i];
                    data[i] = data[j];
                    data[j] = temp;
                }
            }
        }
        return data;
    }
    @Profile(name = "InsertionSort",
             outputs = {jprofile.output.ScatterPlot.class})
    public int[] insertionSort(int[] data) {
        for (int i = 1; i < data.length; i++) {
            int j = i;
            int temp = data[i];
            while (j > 0) {
                //@Profiler.count++;
                if(data[j - 1] < temp){ break;}
                data[j] = data[j - 1];
                j--;
            }
            data[j] = temp;
        }
        return data;
    }
}
```

รหัสที่ 3.7 การกำหนดรูปแบบการทดลองพร้อมกันหลาย ๆ เมท็อด

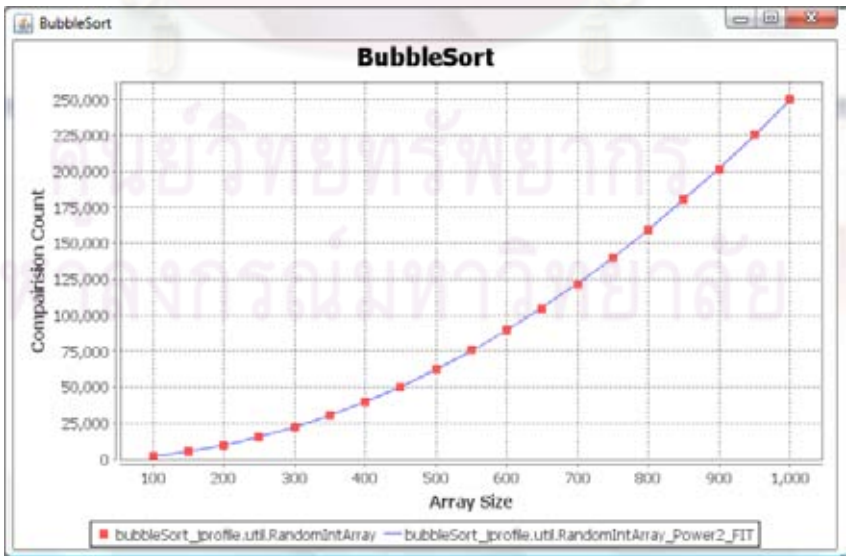
## 3.4 การกำกับ @Profile เพื่อกำหนดรูปแบบการแสดงผล

ผู้ใช้สามารถกำหนด รูปแบบการแสดงผลการทดลองได้โดยการเขียน @Profile กำกับไว้ โดยรายละเอียดของการเขียน @Profile มีดังต่อไปนี้

### 3.4.1 การกำหนดชื่อของขนาดข้อมูลขาเข้า จำนวนการทำงาน และการปรับเส้นโค้งผลลัพธ์

การกำหนดชื่อของขนาดข้อมูลขาเข้า และจำนวนการทำงานให้ตรงตามลักษณะข้อมูลจริง สามารถเพิ่มความเข้าใจต่อผลการทดลองมากขึ้น การแสดงผลการาทดลองในรูปของจุดความสัมพันธ์ระหว่างขนาดของข้อมูลขาเข้า และจำนวนครั้งการทำงาน บางครั้งจะไม่เห็นอัตราเติบโตได้ชัดเจน ผู้ใช้สามารถกำหนดให้ระบบปรับเส้นโค้งที่ผ่านจุดทั้งหลายเพื่อแสดงแนวโน้มการเปลี่ยนแปลงผลลัพธ์ได้ดีขึ้น (ดังตัวอย่างในรูปที่ 3.8) พร้อมทั้งแสดงกราฟเส้นของฟังก์ชั่นที่ได้จากกระบวนการ Curve Fitting ในส่วนคำอธิบายสัญลักษณ์ จะมีข้อมูลแสดงเพิ่มเติมว่า Power2 แสดงว่าข้อมูลผลการทดลองนี้มีอัตราเติบโตแบบ $n^2$ โดยผู้ใช้กำกับข้อมูลเพิ่มเติมใน @Profile ซึ่งแสดงในรหัสที่ 3.8 ประกอบด้วย

- xtitle : ระบุข้อความกำกับแกน X
- ytitle : ระบุข้อความกำกับแกน Y
- curveFitting : ระบุความต้องการให้ระบบแสดงผลลัพธ์ของการปรับเส้นโค้งหรือไม่




รูปที่ 3.8 จำนวนการเปรียบเทียบข้อมูลในการเรียงข้อมูลแบบบับเบิ้ล โดยกำหนดชื่อของขนาดข้อมูลขาเข้าและจำนวนการทำงานพร้อมแสดงผลลัพธ์ของการปรับเส้นโค้ง

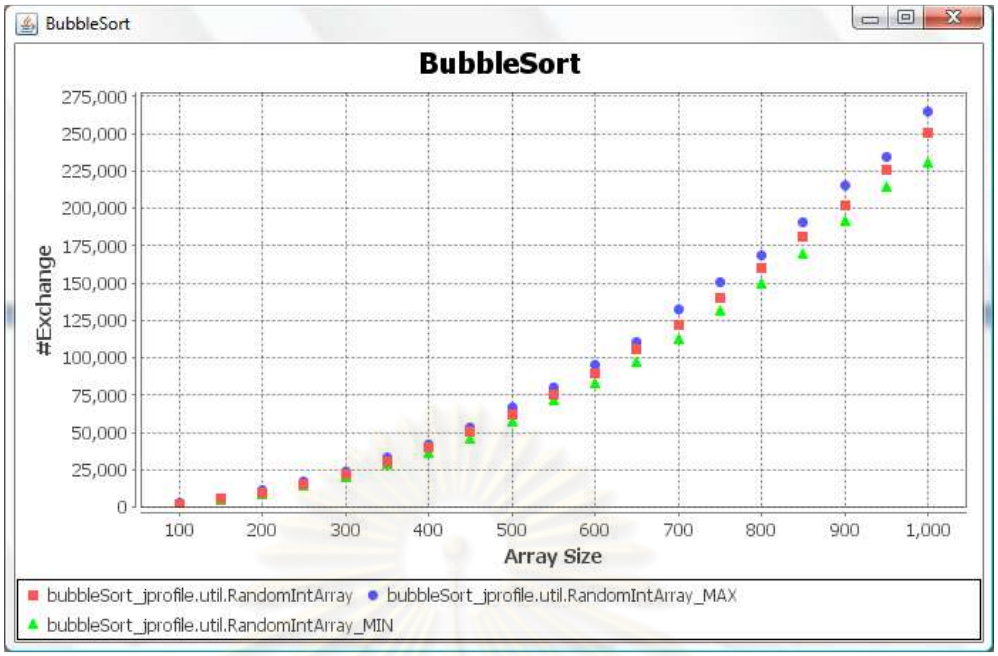
```
...
public class Sort {
    @Profile(name = "BubbleSort",
             xtitle = "Array Size", ytitle = "Compairision Count",
             inputs = {jprofile.util.RandomIntArray.class},
             outputs = {jprofile.output.ScatterPlot.class},
             from = 100, to = 1000, step = 50, repeat = 100,
             curveFitting = true)
    public int[] bubbleSort(int[] data){
        public int[] bubbleSort(int[] data){
        for(int i = 0;i<data.length;i++){
            for(int j = i ;j < data.length ; j++){
                //@Profiler.count++;
                if(data[i] > data[j]){
                    int temp = data[i];
                    data[i] = data[j];
                    data[j] = temp;
                }
            }
        }
        return data;
    }
}
```

รหัสที่ 3.8 การกำหนดรูปแบบการแสดงผลการ โดยกำหนดชื่อข้อมูลขาเข้าและจำนวนครั้งการทำงานพร้อมแสดงผลลัพธ์ของการปรับเส้นโค้ง

### 3.4.3 การแสดงจำนวนครั้งการทำงานมากสุดและน้อยสุดของแต่ละขนาดข้อมูลขาเข้า

ในการศึกษาพฤติกรรมการทำงานของบางอัลกอริทึม นอกจากจำนวนครั้งการทำงานโดยเฉลี่ย อาจต้องการทราบข้อมูลของจำนวนครั้งการทำงานมากสุดและจำนวนครั้งการทำงานน้อยสุด ในรูปที่ 3.9 สนใจจำนวนการสลับข้อมูลในการเรียงข้อมูลแบบบับเบิ้ล โดยเฉลี่ยมากที่สุด และน้อยที่สุด โดยผู้ใช้ต้องเปลี่ยนตำแหน่งของ //@Profiler.count++ ไปยังตำแหน่งก่อนคำสั่งสลับข้อมูล และเพิ่มการกำกับ @Profile ซึ่งแสดงในรหัสที่ 3.9

- displayAvg : ระบุความต้องการให้ระบบแสดงจำนวนการทำงานเฉลี่ยหรือไม่
- displayMax : ระบุความต้องการให้ระบบแสดงจำนวนการทำงานมากสุดหรือไม่ โดยจะแสดงข้อความ MAX เพิ่มเติม ในส่วนคำอธิบายสัญลักษณ์
- displayMin : ระบุความต้องการให้ระบบแสดงจำนวนการทำงานน้อยสุดหรือไม่ โดยจะแสดงข้อความ MIN เพิ่มเติม ในส่วนคำอธิบายสัญลักษณ์



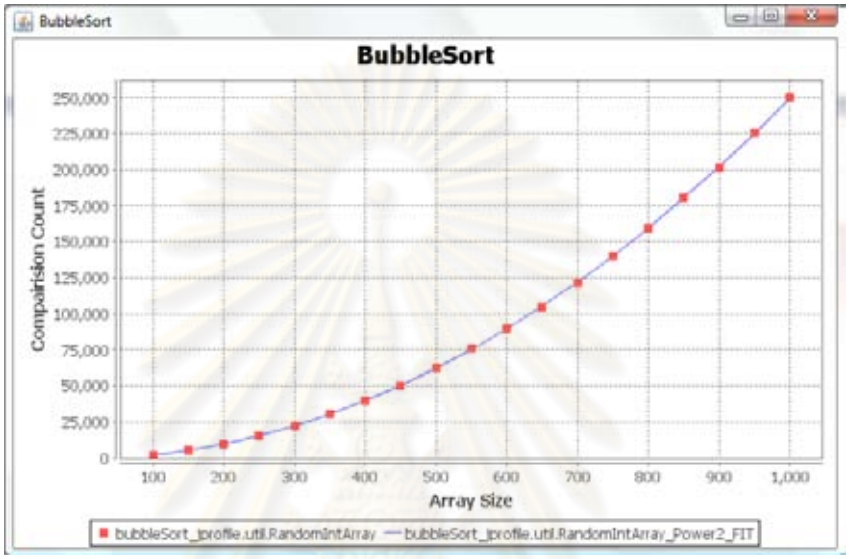

รูปที่ 3.9 จำนวนการสลับข้อมูลในการเรียงข้อมูลแบบบับเบิ้ล โดยเฉลี่ย, มากที่สุด และน้อยที่สุด สัมพันธ์กับขนาดข้อมูลขาเข้า

```
...
public class Sort {
    @Profile(name = "BubbleSort",
             xtitle = "Array Size", ytitle = "#Exchange",
             inputs = { jprofile.util.RandomIntArray.class },
             outputs = {jprofile.output.ScatterPlot.class},
             from = 100, to = 1000, step = 50, repeat = 100,
             displayAvg = true, displayMax = true, displayMin = true)
    public int[] bubbleSort(int[] data){
        for(int i = 0;i<data.length;i++){
            for(int j = i ;j < data.length ; j++){
                if(data[i] > data[j]){
                    //@Profiler.count++;
                    ...   //Swap
                }
            }
        }
        return data;
    }
}
```

รหัสที่ 3.9 การกำหนดรูปแบบการทดลองให้แสดงจำนวนครั้งการทำงานมากสุดและน้อยสุด

### 3.4.4 การแสดงผลการทำลองหลายรูปแบบ

ในการวิเคราะห์เชิงทดลอง ที่ต้องการให้แสดงผลการทดลองหลายรูปแบบ ดังรูปที่ 3.10 และ รูปที่ 3.11 โดยใช้หน่วยแสดงผลการทดลองแบบต่าง ๆ ผู้ใช้สามารถเพิ่มการกำกับ @Profile ในส่วน outputs ในที่นี้กำหนดให้แสดงผลการทดลองเป็น กราฟและไฟล์ Excel ดังแสดงในรหัสที่ 3.10




รูปที่ 3.10 กราฟแสดงจำนวนการสลับข้อมูลในการเรียงข้อมูลแบบบับเบิ้ล พร้อมแสดงผลลัพธ์ของการปรับเส้นโค้ง

| | A | B | C |
|---|---|---|---|
| 1 | Array Size | bubbleSort_jprofile.util.RandomIntArray | bubbleSort_jprofile.util.RandomIntArray_Power2_FIT |
| 2 | 100 | 2480.15 | 2475.519507 |
| 3 | 150 | 5575.16 | 5597.726255 |
| 4 | 200 | 9908.31 | 9968.815702 |
| 5 | 250 | 15670.63 | 15588.78785 |
| 6 | 300 | 22409.86 | 22457.64269 |
| 7 | 350 | 30419.07 | 30575.38024 |
| 8 | 400 | 40038.7 | 39942.00048 |
| 9 | 450 | 50608 | 50557.50342 |
| 10 | 500 | 62278.38 | 62421.88907 |
| 11 | 550 | 75252.02 | 75535.15741 |
| 12 | 600 | 90049.36 | 89897.30845 |
| 13 | 650 | 105611.5 | 105508.3422 |
| 14 | 700 | 122734.48 | 122368.2586 |
| 15 | 750 | 140472.48 | 140477.0578 |
| 16 | 800 | 159718.51 | 159834.7396 |
| 17 | 850 | 180746.28 | 180441.3041 |
| 18 | 900 | 202293.25 | 202296.7514 |
| 19 | 950 | 225188.97 | 225401.0813 |
| 20 | 1000 | 249644.45 | 249754.2939 |
| 21 | | | |

BubbleSort

Ready

รูปที่ 3.11 ไฟล์ Excel แสดงจำนวนการสลับข้อมูลในการเรียงข้อมูลแบบบับเบิ้ล พร้อมแสดงผลลัพธ์ของการปรับเส้นโค้ง

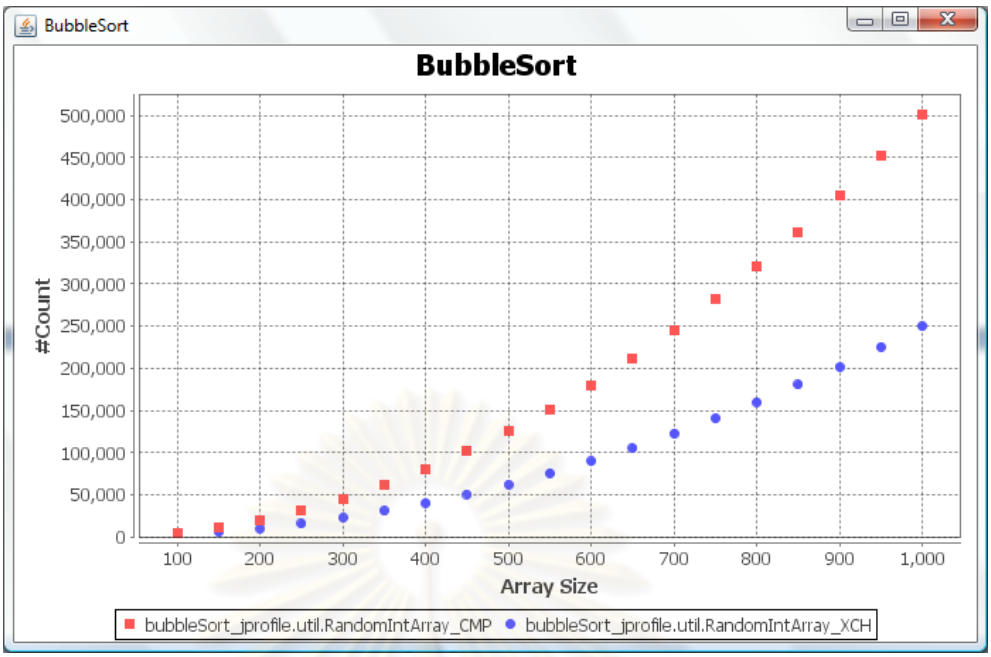
```
...
public class Sort {
    @Profile(name = "BubbleSort",
    xtitle = "Array Size", ytitle = "#Exchange",
    inputs = {jprofile.util.RandomIntArray.class},
    outputs = {jprofile.output.ScatterPlot.class,
               jprofile.output.ExcelFile.class},
    from = 100, to = 1000, step = 50, repeat = 100,
    curveFitting = true)
    public int[] bubbleSort(int[] data){
        for(int i = 0;i<data.length;i++){
            for(int j = i ;j < data.length ; j++){
                if(data[i] > data[j]){
                    //@Profiler.count++;
                    int temp = data[i];
                    data[i] = data[j];
                    data[j] = temp;
                }
            }
        }
        return data;
    }
}
```

รหัสที่ 3.10 การกำหนดรูปแบบการทดลองให้แสดงผลการทดลองหลายรูปแบบ

### 3.4.5 การกำหนดชื่อให้ตัวนับ

ในการวิเคราะห์เชิงทดลองที่ใช้ตัวนับหลายตัว อาจเกิดความสับสนว่าตัวนับแต่ละตัวเก็บข้อมูลอะไร ทั้งในขั้นตอนการเขียนรหัสต้นฉบับ และการแสดงผลการทดลอง ดังรูปที่ 3.5 และ รหัสที่ 3.6 ผู้ใช้สามารถกำหนดชื่อให้ตัวนับแต่ละตัวได้ ดังแสดงในรูปที่ 3.12 ในส่วนคำอธิบายสัญลักษณ์จะเปลี่ยนหมายเลขตัวนับเป็นชื่อตัวนับที่กำหนดไว้แทน โดยเพิ่มการกำกับ @Profile ซึ่งแสดงในรหัสที่ 3.11 ประกอบด้วย

- countNames : ระบุชื่อของตัวนับแต่ละตัว ในที่นี่กำหนดให้ตัวนับตัวที่หนึ่งนับจำนวนการเปรียบเทียบ (CMP) และ ตัวนับตัวที่สองนับจำนวนการสลับตำแหน่ง (XCH) เมื่อมีการกำหนด ค่า countNames แล้ว ผู้ใช้ไม่จำเป็นต้องกำหนด ค่า countNo เพื่อระบุจำนวนตัวนับที่ใช้ เพราะตัวควบคุมการทดลองใช้ขนาดของ countNames ในการทดลอง




รูปที่ 3.12 จำนวนการเปรียบเทียบและการสลับข้อมูลในการเรียงข้อมูลแบบบับเบิ้ล พร้อมกำหนดชื่อให้ตัวนับ

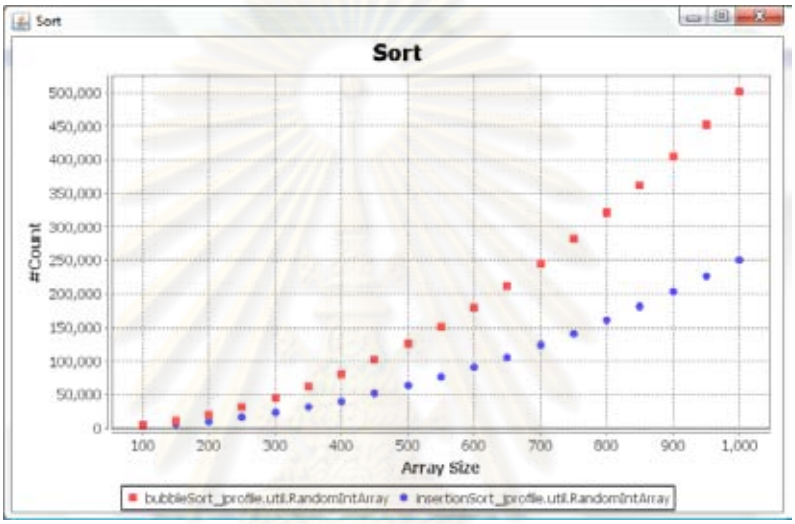
```
...
public class Sort {
    @Profile(
              ...
              countNames = {"CMP", "XCH"})
    public int[] bubbleSort(int[] data) {
        for (int i = 0; i < data.length; i++) {
            for (int j = i; j < data.length; j++) {
                //@Profiler.counts[0]++;
                if (data[i] > data[j]) {
                    //@Profiler.counts[1]++;
                    int temp = data[i];
                    data[i] = data[j];
                    data[j] = temp;
                }
            }
        }
        return data;
    }
}
```

รหัสที่ 3.11 การกำหนดรูปแบบการทดลองโดยกำหนดชื่อให้ตัวนับ

### 3.4.6 การกำหนดให้รวมการแสดงผลเมื่อ ทำการทดลองหลายเมท็อด

ในการวิเคราะห์เชิงทดลองที่ต้องการเปรียบเทียบการทำงานของหลาย ๆ เมท็อด แล้วต้องการให้รวมการแสดงผลไว้ในหน่วยแสดงผลตัวเดียวกัน ดังรูปที่ 3.13 ผู้ใช้สามารถเพิ่มการกำกับ @Profile ดังแสดงในรหัสที่ 3.12 ซึ่งประกอบด้วย

- outputs : กำหนดหน่วยแสดงผล ถ้ากำหนดค่าไว้ที่ระดับคลาสแทนระดับเมท็อด จะเป็นการกำหนดให้รวมการแสดงผลการทดลองของทุกเมท็อดไว้ด้วยกัน



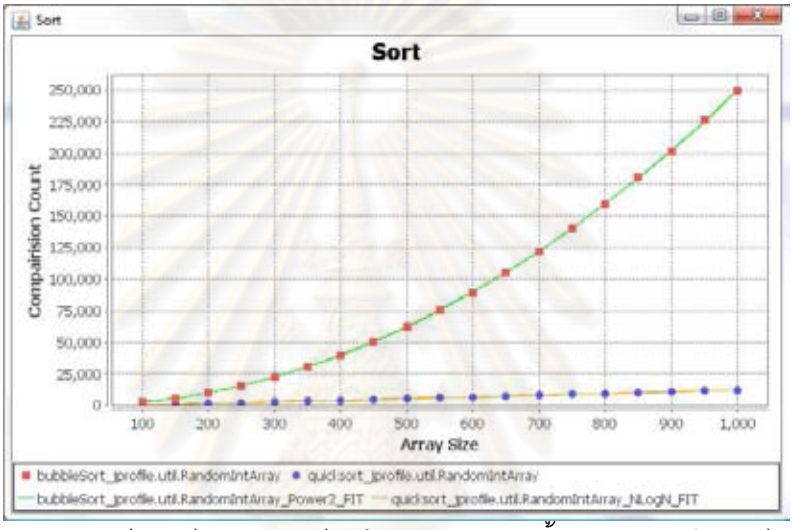

รูปที่ 3.13 จำนวนการเปรียบเทียบในการเรียงข้อมูลแบบบับเบิ้ล และแบบแทรกใน หน่วยแสดงผลเดียว

```
...
@Profile(
         ...
         outputs = {jprofile.output.ScatterPlot.class},
         seed = 1000L)
public class Sort {
    @Profile()
    public int[] bubbleSort(int[] data) {
        ...
    }
    @Profile()
    public int[] insertionSort(int[] data) {
        ...
    }
}
```

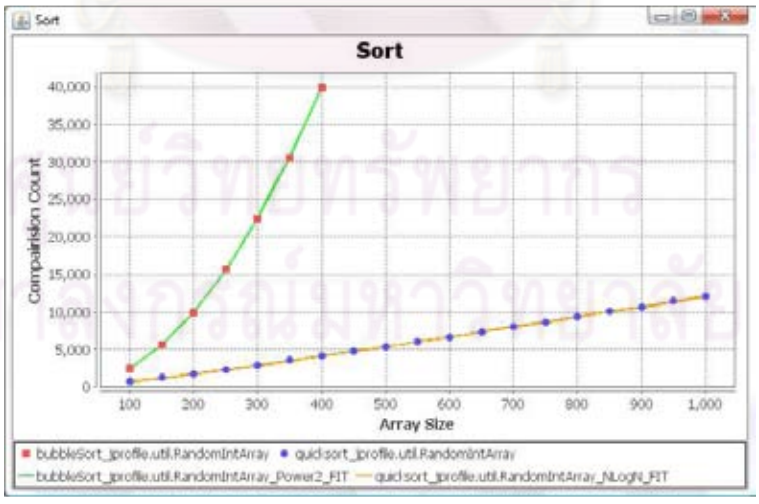
รหัสที่ 3.12 การกำหนดรูปแบบการทดลองพร้อมกันหลายเมท็อดแบบรวมการแสดงผล

### 3.4.7 การกำหนดค่า Y มากสุดในการแสดงผล

เมื่อมีการใช้ตัวนับหลายตัว หรือกำหนดให้แสดงผลการทดลองของหลายเมท็อดในหน่วยแสดงผลเดียวกัน อาจเป็นไปได้ที่ผลการทดลองสองชุดนั้นมีความแตกต่างกันมาก ทำให้ไม่เห็นรายละเอียดของชุดผลการทดลองที่มีค่าน้อย ดังรูปที่ 3.14 จะมองเห็นว่าจำนวนครั้งการเปรียบเทียบข้อมูลในการเรียงข้อมูลแบบเร็ว เป็นเส้นตรงขนานแกน X แต่ผู้ใช้สามารถกำหนดค่า Y มากสุดในการแสดงผล เพื่อให้เห็นรายละเอียดของจำนวนครั้งการเปรียบเทียบข้อมูลในการเรียงข้อมูลแบบเร็ว ดังรูปที่ 3.15 โดยเพิ่มการกำหนดค่า maxY ใน @Profile ดังแสดงในรหัสที่ 3.13




รูปที่ 3.14 จำนวนการเปรียบเทียบในการเรียงข้อมูลแบบบับเบิ้ล และแบบเร็ว โดยไม่กำหนดค่า Y มากสุด




รูปที่ 3.15 จำนวนการเปรียบเทียบในการเรียงข้อมูลแบบบับเบิ้ล และแบบเร็ว โดยกำหนดค่า Y มากสุด

```
@Profile(
         ...
         maxY=50000
)
public class Sort {
    @Profile()
    public int[] bubbleSort(int[] data) { ... }
    @Profile()
    public void quicksort(int[] a) {
        quicksort(a, 0, a.length - 1);
    }
    public void quicksort(int[] a, int left, int right) {
        if (right <= left) return;
        int i = partition(a, left, right);
        quicksort(a, left, i - 1);
        quicksort(a, i + 1, right);
    }
    private int partition(int[] a, int left, int right) {
        int i = left - 1;
        int j = right;
        while (true) {
            while (less(a[++i] , a[right]));
            while (less(a[right] , a[--j])) {
                if (j == left) break;
            }
            if (i >= j) break;
            exch(a, i, j);
        }
        exch(a, i, right);
        return i;
    }
    private  boolean less(int x, int y) {
        //@Profiler.count++;
        return (x < y);
    }
    private void exch(int[] a, int i, int j) {
        int swap = a[i];
        a[i] = a[j];
        a[j] = swap;
    }
}
```

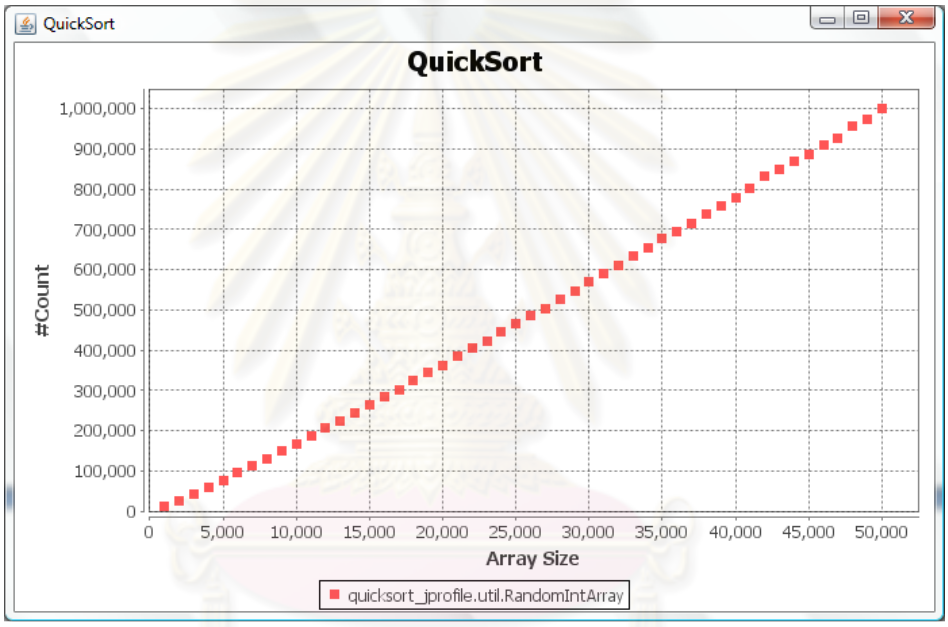
รหัสที่ 3.13 การกำหนดค่า Y มากสุดในการแสดงผลการทดลอง

### 3.4.8 การปรับข้อมูลผลการทดลอง และการใช้ผลการทดลองเดิม

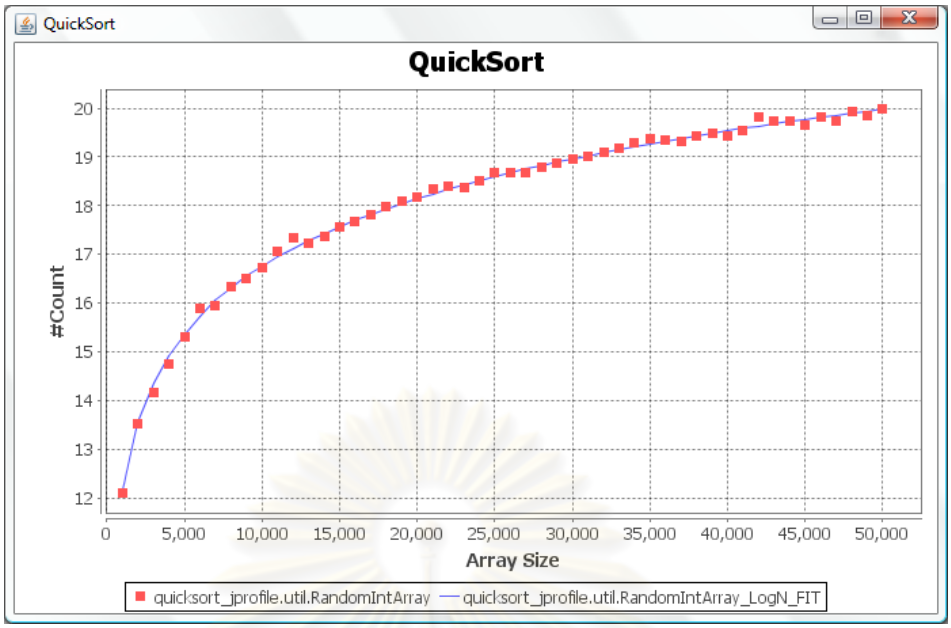
ในการวิเคราะห์อัลกอริทึมการเรียงข้อมูลเร็ว โดยสนใจจำนวนครั้งการเปรียบเทียบข้อมูล จะได้กราฟที่มีอัตราการเติบโตเป็นเส้นตรง ดังรูปที่ 3.16 ยังไม่แน่ใจว่ามีอัตราการเติบโตแบบ $O(n \log n)$ ตามการวิเคราะห์เชิงคณิตศาสตร์จริงหรือไม่ แต่เมื่อปรับผลการทดลองโดยการหารด้วยจำนวนข้อมูล $(n)$ จะได้กราฟที่มีอัตราการเติบโตแบบ $O(\log n)$ ดังรูปที่ 3.17 ทำให้มั่นใจได้ว่าผลการทดลองก่อนปรับนั้น มีอัตราการเติบโตแบบ $O(n \log n)$ ตามการ

วิเคราะห์เชิงคณิตศาสตร์จริง และเนื่องจากในการวิเคราะห์เชิงทดลองบางครั้งจะใช้เวลานานมาก เพื่อให้การปรับข้อมูลผลการทดลองรวมถึงรูปแบบการแสดงผลการทดลองอื่น ๆ ไม่ต้องทำการทดลองใหม่ ซึ่งจะช่วยลดเวลาในการวิเคราะห์ ระบบจะเก็บข้อมูลผลการทดลองไว้และนำมาใช้เมื่อผู้ใช้ต้องการ ผู้ใช้สามารถกำหนดเมท็อดปรับข้อมูล และกำหนดให้ใช้ผลการทดลองเดิม โดยเพิ่มการกำกับ @Profile และเขียนเมท็อดปรับข้อมูล ดังแสดงในรหัสที่ 3.14 ซึ่งประกอบด้วย

- adjustFunction : ระบุชื่อของเมท็อดปรับค่าสำหรับตัวนับแต่ละตัว ซึ่งเป็นเมท็อดที่กำหนดให้รับพารามิเตอร์เป็นข้อมูลชนิด double สองตัว และคืนข้อมูลชนิด double เช่นกัน
- useOldData : ระบุความต้องการให้ระบบใช้ผลการทดลองเดิมหรือไม่




รูปที่ 3.16 จำนวนการเปรียบเทียบในการเรียงข้อมูลแบบเร็ว




รูปที่ 3.17 จำนวนการเปรียบเทียบในการเรียงข้อมูลแบบเร็ว โดยมีการปรับข้อมูลผลการทดลองและใช้ข้อมูลผลการทดลองเดิม

```
...
public Class Sort {
    @Profile(name = "QuickSort",
             xtitle = "Array Size", ytitle = "#Count",
             inputs = {jprofile.util.RandomIntArray.class},
             outputs = {jprofile.output.ScatterPlot.class},
             from = 1000, to = 50000, step = 1000, repeat = 100,
             adjustFunctions={"divideByN"}, useOldData=true,
             curveFitting=true)
    public int[] quicksort(int[] data) {
       ...
    }
    public double divideByN (double x ,double y){
        return y / x;
    }
```

รหัสที่ 3.14 การปรับข้อมูลผลการทดลอง และใช้ผลการทดลองเดิม

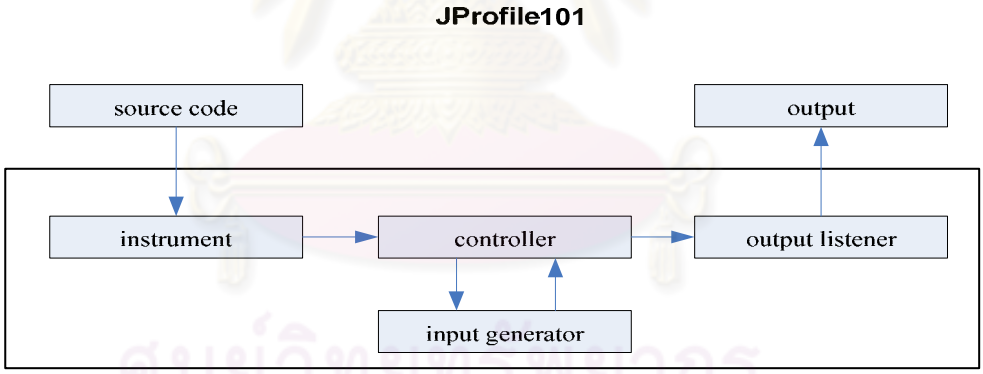
# บทที่ 4

# ตัวควบคุมและคลาสอรรถประโยชน์สำหรับการวิเคราะห์อัลกอริทึมเชิงทดลอง

งานวิจัยนี้ศึกษา ออกแบบ และพัฒนาตัวควบคุมและคลาสอรรถประโยชน์สำหรับการวิเคราะห์อัลกอริทึมเชิงทดลองซึ่งเรียกว่า “JProfile101” ตัวควบคุมและคลาสอรรถประโยชน์นี้อำนวยความสะดวกในการผลิตข้อมูลขาเข้า เก็บข้อมูลระหว่างการทำงานของโปรแกรม และการแสดงผลลัพธ์ เพื่อเสริมความเข้าใจในการศึกษาพฤติกรรม และประสิทธิภาพการทำงานของอัลกอริทึม ตัวควบคุมอาศัยการแทรกคำสั่งนับ ทำให้โปรแกรมที่ทดลองทำงานได้เต็มความเร็ว จึงสามารถทดลองกับปริมาณข้อมูลที่ขนาดที่โปรแกรมนั้นทำงานจริง

## 4.1 ภาพรวมของระบบ

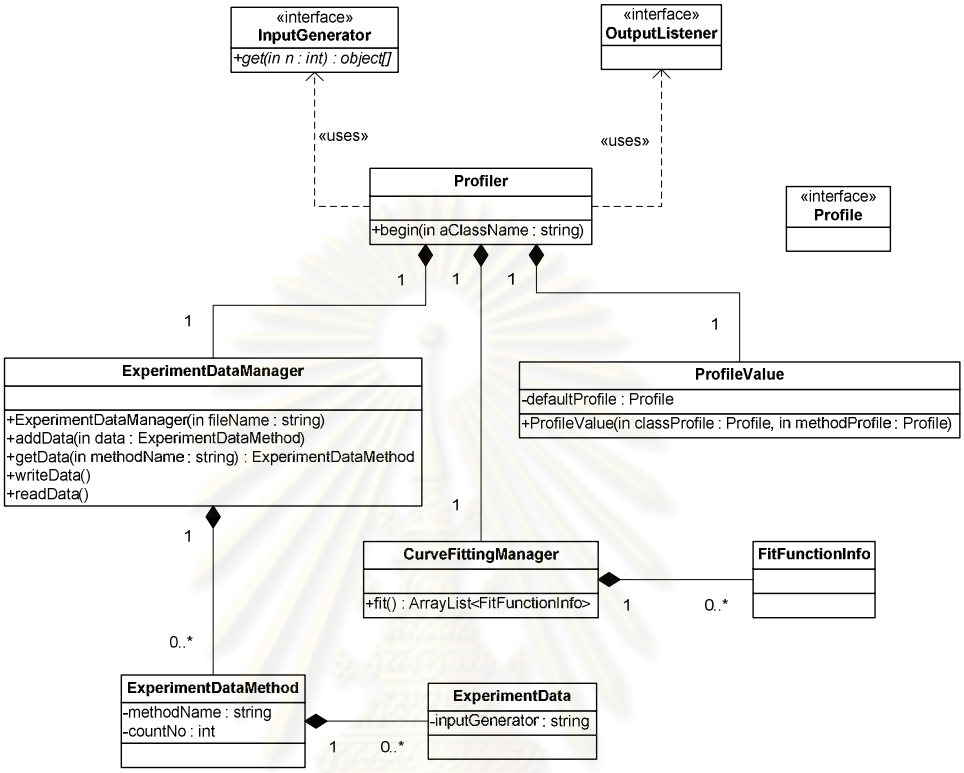
กระบวนการวิเคราะห์อัลกอริทึมเชิงทดลอง แบ่งออกเป็น 6 ส่วน คือ รหัสต้นฉบับของโปรแกรมที่ต้องการวิเคราะห์ (source code), ตัวเปลี่ยนรหัสต้นฉบับ (instrument), ตัวควบคุม (controller), ตัวผลิตข้อมูลขาเข้าสำหรับการทดลอง (input generator), หน่วยแสดงผล (output listener) และผลการทดลอง (output) ดังแสดงในรูปที่ 4.1




รูปที่ 4.1 โครงสร้างของระบบ JProfile101

เมื่อผู้ใช้ต้องการวิเคราะห์อัลกอริทึมโดยใช้ระบบ JProfile101 ต้องเริ่มด้วยการเขียนโปรแกรมตามอัลกอริทึมที่ต้องการวิเคราะห์ จากนั้นเขียนลักษณะของการทดลองกำกับเป็น annotation ที่หัวเมท็อดหรือหัวคลาสและแทรกรหัสคำสั่งเก็บข้อมูลไว้ตำแหน่งที่ต้องการ เมื่อสั่งให้ตัวควบคุมทำงาน ตัวควบคุมจะแปลงรหัสต้นฉบับและเริ่มทดลองตามที่ได้เขียนกำกับไว้

จากโครงสร้างของระบบ ได้มีการออกแบบรายละเอียดของแต่ละส่วนของระบบ โดยใช้แผนภาพคลาส รูปที่ 4.2




รูปที่ 4.2 แผนภาพคลาสของระบบ JProfile101

## 4.2 **คลาสอรรถประโยชน์**

ในการเรียนการสอนเนื้อหาที่เกี่ยวกับการวิเคราะห์อัลกอริทึม มักจะเป็นการวิเคราะห์การทำงานของอัลกอริทึมกับข้อมูลขาเข้า ว่ามีการอัตราการเติบโตของฟังก์ชันเป็นอย่างไร เมื่อใช้ระบบช่วยวิเคราะห์อัลกอริทึมเชิงทดลอง ตัวควบคุมก็จะทำงานโดยนำข้อมูลขาเข้าขนาดต่าง ๆ ตามที่กำหนดไว้ส่งให้โปรแกรมที่เขียนตามอัลกอริทึมที่สนใจทำงาน และเก็บข้อมูลการทำงานของคำสั่งที่กำหนดไว้ จากนั้นนำข้อมูลที่ได้ไปแสดงผล เนื่องจากข้อมูลขาเข้า และการแสดงผลมีรูปแบบมากมายหลายลักษณะ จึงจำเป็นต้องมีคลาสอรรถประโยชน์เพื่อผลิตข้อมูลขาเข้าและแสดงผลให้กับตัวควบคุม เป็นหน่วยผลิตข้อมูลขาเข้า และหน่วยแสดงผล

### 4.2.1 หน่วยผลิตข้อมูลขาเข้า

หน่วยผลิตข้อมูลขาเข้า เป็นคลาสอรรถประโยชน์ที่ทำหน้าที่ผลิตข้อมูลขาเข้าที่มีขนาดและลักษณะตามที่ตัวควบคุมต้องการ

หน่วยผลิตข้อมูลขาเข้าเป็นคลาสที่ implements อินเทอร์เฟซ `InputGenertor` ดังแสดงในรหัสที่ 4.1 ซึ่งบังคับให้เขียนเพียงหนึ่งเมท็อดคือ `Object[] get(int n)` มีหน้าที่ผลิตข้อมูลขนาด n ซึ่งอาเรย์ของ `Object` ที่หน่วยผลิตข้อมูลขาเข้าคืนค่าให้ตัวควบคุมนี้จะถูกใช้เป็นพารามิเตอร์ในการเรียกโปรแกรมทำงาน ที่รหัสที่ 4.2 แสดงตัวอย่างคลาสสำหรับสร้างหน่วยผลิตอาเรย์ของจำนวนเต็มขนาด n ช่องเก็บค่าสุ่ม

```
public interface InputGenerator {
    public Object[] get(int n);
}
```

รหัสที่ 4.1 อินเทอร์เฟซ InputGenerator

```
public class RandomIntArray implements InputGenerator{
    public Object[] get(int n) {
        int[] d = new int[n];
        for(int i = 0 ; i < d.length ; i++){
            d[i] = (int) (100000000*Math.random());
        }
        return new Object[]{d};
    }
}
```

รหัสที่ 4.2 หน่วยผลิตอาเรย์ของจำนวนเต็มขนาด n ช่องเก็บค่าสุ่ม

ตารางที่ 4.1 แสดงตัวอย่างคลาสหน่วยผลิตข้อมูลขาเข้า ซึ่งระบบ JProfile101 ได้จัดเตรียมไว้อำนวยความสะดวกสำหรับผู้วิเคราะห์ใช้กำหนดให้กับตัวควบคุม ตามลักษณะข้อมูลขาเข้าที่พบบ่อยในการเรียนการสอนเนื้อหาที่เกี่ยวกับการวิเคราะห์อัลกอริทึม

| หน่วยผลิตข้อมูลขาเข้า | รายละเอียด |
|---|---|
| SortedIntArray | ผลิตข้อมูลในอาเรย์แบบจำนวนเต็มเรียงลำดับจากน้อยไปหามาก |
| RandomIntArray | ผลิตข้อมูลในอาเรย์แบบจำนวนเต็มมีค่าสุ่ม |
| ReverseIntArray | ผลิตข้อมูลในอาเรย์แบบจำนวนเต็มเรียงลำดับจากมากไปหาน้อย |

ตารางที่ 4.1 หน่วยผลิตข้อมูลขาเข้า

### 4.2.2 **หน่วยแสดงผล**

หน่วยแสดงผล เป็นคลาสอรรถประโยชน์ที่ทำหน้าที่คอยรับข้อมูลการทำงานของโปรแกรมจากตัวควบคุม เมื่อเกิดการเปลี่ยนแปลงที่ส่งผลกระทบกับหน่วยแสดงผล แล้วนำข้อมูลนั้นไปแสดงผล หน่วยแสดงผลเป็นคลาสที่ implement อินเทอร์เฟซ `OutputListener` ซึ่งบังคับให้เขียนเมท็อด ดังที่แสดงในรหัสที่ 4.3 และมีรายละเอียดดังแสดงในตารางที่ 4.2

```
public interface OutputListener {
    public void name(String name);
    public void xtitle(String xtitle);
    public void ytitle(String ytitle);
    public void newInput(String inputGeneratorName);
    public void newMethod(String methodName, int no,String[] countNames);
    public void newData(double n, double[] avg, double[] max, double[] min);
    public void maxY(double value);
    public void finish();
    public void fitFunction(ArrayList<FitFunctionInfo> fitFunctionList);
    public void timer(double time);
    public void commonOutput(boolean commonOutput);
}
```

รหัสที่ 4.3 อินเทอร์เฟซ OutputListener

| เมท็อด | รายละเอียด |
|---|---|
| name(String name) | รับชื่อของหน่วยแสดงผล |
| xtitle(String xtitle) | รับข้อความกำกับแกน X |
| ytitle(String ytitle) | รับข้อความกำกับแกน Y |
| newInput(String inputGeneratorName) | รับชื่อหน่วยผลิตข้อมูลขาเข้า เมื่อมีการเปลี่ยนหน่วยผลิตข้อมูลขาเข้า |
| newMethod(String methodName, int no,String[] countNames) | รับชื่อเมท็อด, จำนวนตัวนับ และชื่อตัวนับ เมื่อมีการเปลี่ยนเมท็อด |
| newData(double n, double[] avg, double[] max, double[] min); | รับข้อมูลการทดลองประกอบด้วย ขนาดข้อมูล , จำนวนครั้งการทำงานเฉลี่ย,จำนวนครั้งการทำงานมากสุด และจำนวนครั้งการทำงานน้อยสุด |
| maxY(double value) | รับค่า Y มากสุดในการแสดงผล |

| | |
|---|---|
| finish() | ถูกเรียกเมื่อการทดลองเสร็จ ทำหน้าที่แสดงผลการทดลอง |
| fitFunction(ArrayList<FitFunctionInfo> fitFunctionList) | รับรายการของ ฟังก์ชั่นปรับเส้นโค้งผลลัพธ์ |
| timer(Double time) | รับเวลาการทำงานโดยเฉลี่ย ที่คิดจากข้อมูลที่ลบข้อมูลค่ามาก และค่าน้อยตามที่กำหนด |
| commonOutput( boolean commonOutput) | รับเงื่อนไขการแสดงผลว่าให้แสดงผลลัพท์ของเมท็อดไว้ด้วยกันหรือไม่ |

ตารางที่ 4.2 รายละเอียดเมท็อดของอินเทอร์เฟซ OutputListener

### 4.3 ตัวควบคุม

การวิเคราะห์เชิงทดลองดำเนินการภายใต้การทำงานของตัวควบคุม ซึ่งถูกสั่งให้ทำงานด้วยคำสั่ง

```
new Profiler().begin(classname);
```

สิ่งที่ส่งให้ตัวควบคุมคือชื่อคลาส ซึ่งภายในมีเมท็อดที่ถูกกำกับด้วย annotation `@Profile` เป็นข้อมูลที่ผู้วิเคราะห์กำหนดให้กับตัวควบคุมใช้ในการทดลอง ลักษณะการทดลอง เช่น หน่วยผลิตข้อมูลขาเข้า, ปริมาณข้อมูลขาเข้า, จำนวนครั้งในการทำการทดลองซ้ำ และจำนวนตัวนับที่ใช้ เป็นต้น รวมถึงการแสดงผล ได้แก่ แสดงจำนวนครั้งการทำงานโดยเฉลี่ย, แสดงเวลาที่ใช้โดยเฉลี่ย, ร้อยละของเวลาที่ไม่นำมาคิดค่าเฉลี่ย การแสดง curve fittings, การกำหนดค่า Y มากสุด, การใช้ผลการทดลองเดิม และ รายชื่อเมท็อดสำหรับปรับข้อมูลผลการทดลอง การทำงานของตัวควบคุม แบ่งออกเป็น 2 ลักษณะ คือ แบบทำการทดลองจริง และ แบบแสดงผลการทดลองเดิมในรูปแบบใหม่ รหัสที่ 4.4 แสดง รหัสเทียมการทำงานของตัวควบคุมกรณีทำการทดลองจริง มีลำดับการทำงานดังนี้

```
01:  public class Profiler {
02: void begin(clsName) {
03:  init();
04:  instrumentation(clsName);
05:  classProf = cls.getProfileAnnotation();
06:  expDataMgr = createExpDataMgr(className);
07:  //-----------------------
08:  for (method : cls.getProfiledMethods()) {
09:   methodProf = method.getProfileAnnotation();
```

```
10:    profValue.extractProfile(classProf, methodProf);
11:    addOutputListeners(profValue);
12:    countNames  = createCountNames(profValue, countNo);
13:    adjustMethods  = createAdjustMethods(profValue, countNo);
14:    expDataMethod = ExperimentDataMethod(method.getName(), countNo);
15:    random = createRandom(profValue.seed());
16:    fireEvent(Event.METHOD, method.getName(), countNo, countNames);
17:     //----------------------
18:    for (input : profValue.inputs()) {
19:     fireEvent(Event.INPUT, input);
20:     expData = expDataMethod.createExpData(input.getName());
21:     ingen = input.createInstance();
22:     fitMgr.newInput(ingen);
23:     //----------------------
24:     for(n=profValue.from();n<=profValue.to();n+=profValue.step()) {
25:      resetSumCounts();
26:      for (k = 0; k < profValue.repeat(); k++) {
27:       resetCounts();
28:       data = ingen.getData(n);
29:       stratTime = System.nanoTime();
30:       method.invoke(data);
31:       endTime = System.nanoTime();
32:       times[k] = endTime - startTime;
33:       adjCount(sumCounts);
34:       getCount(sumAdjCounts);
35:      }
36:      double[] avg = avgCounts(sumAdjCounts);
37:      fitMgr.newData(n,avg);
38:      fireEvent(Event.DATA, n, avg);
39:      time = avgTime(times, profValue.timeTruncate());
40:      fireEvent(Event.TIME, n, time);
41:      getExpData();
42:     }
43:     expDataMgr.addData(expDataMethod)
44:     fireEvent(Event.FIT, n, fitMgr.fit());
45:    }
46:   }
47:   expDataMgr.writeExperimentData();
48:   fireEvent(Event.FINISH,null);
49:   restoreSourceFile();
50:  }
51:…
```

รหัสที่ 4.4 รหัสเทียมของตัวควบคุมกรณีทำการทดลองจริง

- `init` (บรรทัดที่ 3) ทำหน้าที่อ่านค่าตำแหน่งของรหัสต้นฉบับ และตำแหน่งที่ให้บันทึกแฟ้มคลาสใหม่ไปไว้ ซึ่งผู้วิเคราะห์กำหนดไว้ จากไฟล์ชื่อ profile.properties
- `instrumentation` (บรรทัดที่ 4) ทำหน้าที่สำเนารหัสต้นฉบับ จากนั้นเปลี่ยนคำสั่งปรับค่าตัวนับที่เขียนเป็นหมายเหตุ เช่น `//@Profiler.count++` ให้เป็น `Profiler.count++` เรียกใช้ Java Compiler API [12] เพื่อสร้างแฟ้มคลาสใหม่ แล้วบันทึกไปยังตำแหน่งที่กำหนดไว้

- `getProfileAnnotation` (บรรทัดที่ 5) ทำหน้าที่อ่านตัวกำกับ รูปแบบการทดลอง และรูปแบบการแสดงผลของคลาส

- `getProfiledMethods` (บรรทัดที่ 8) ทำหน้าที่ดึงเมท็อดที่ถูกกำกับด้วย annotation `@Profile` มาสั่งทำงานทีละเมท็อดด้วยวงวน การหาเมท็อดในลักษณะนี้อาศัย Java Reflection API [13]

- `getProfileAnnotation` (บรรทัดที่ 9) ทำหน้าที่อ่านตัวกำกับ รูปแบบการทดลอง และรูปแบบการแสดงผลของเมท็อด

- `extractProfile` (บรรทัดที่ 10) ทำหน้าที่เปรียบเทียบค่าระหว่างตัวกำกับของคลาส ตัวกำกับของเมท็อด และ ค่าเริ่มต้นของตัวกำกับแล้วรวบรวมให้ ดังที่แสดงในรหัสที่ 4.5

```
void extractProfile(Profile classProf, Profile methodProf) {
  if (classProf != null) {
      name = classProf.name();
      ...
      if (methodProf != null) {
        if (!methodProf.name().equalsIgnoreCase(defaultProf.name())){
            name = methodProf.name();
        }
        ...
      }
  } else if (methodProf!= null) {
      name = methodProf.name();
      ...
  }
}
```

รหัสที่ 4.5 รหัสเทียมการเปรียบเทียบค่าตัวกำกับ 3 ประเภท

- ตัวควบคุมอ่านรายการของหน่วยแสดงผลจาก `profileVale` ส่งไปลงทะเบียนด้วย `addOutputListeners` (บรรทัดที่ 11) เมื่อเกิดการเปลี่ยนแปลงที่ส่งผลกระทบกับหน่วยแสดงผล เช่น ได้ข้อมูลใหม่จากการทดลอง ตัวควบคุมจะแจ้งให้หน่วยแสดงผลต่าง ๆ รับทราบด้วยการเรียกเมท็อด `fireEvent` ดังที่แสดงใน รหัสที่ 4.6

```
void addOutputListeners(Profile an) {
  for(output : an.outputs())
    listeners.add(createInstance(output, an));
}
void fireEvent(Event evtType, Object args...) {
  for(t : listeners) {
    switch (evtType) {
      case INPUT:
        t.newInput((String) args[0]); break;
      case DATA:
        double n = (Double) args[0];          //n
        double[] avgs = null;                 //avg
        if (args[1] != null) {
```

```
            Double[] tempArray = (Double[])args[1];
            avgs = new double[tempArray.length];
            for(int ind = 0;ind < tempArray.length ; ind++){
                   avgs[ind] = tempArray[ind];
            }
          }
          double[] maxs = null;                //max
          if (args[2] != null) {
               Double[] tempArray = (Double[])args[2];
               maxs = new double[tempArray.length];
               for(int ind = 0;ind < tempArray.length ; ind++){
                    maxs[ind] = tempArray[ind];
               }
          }
          double[] mins = null;                //min
          if (args[3] != null) {
                Double[] tempArray = (Double[])args[3];
                mins = new double[tempArray.length];
                for(int ind = 0;ind < tempArray.length ; ind++){
                     mins[ind] = tempArray[ind];
                }
          }
          t.newData(n, avgs, maxs, mins); break;
        case MAX:
          t.newMax((Double) args[0]); break;
        case MIN:
          t.newMin((Double) args[0]); break;
        case FIT:
          t.fitFunction((ArrayList<FitFunctionInfo>) args[0]); break;
        case TIMER:
          double timer = (Double) args[0];
          t.timer(timer); break;
        case METHOD:
          int countNo = (Integer) args[1];
          t.newMethod((String) args[0], countNo, (String[]) args[2]); break;
        case FINISH:
          t.finish(); break;
        default:
          assert false;
      }
    }
}
```

รหัสที่ 4.6 รหัสเทียมการเพิ่มหน่วยแสดงผลและการแจ้งเหตุการให้หน่วยแสดงผล

- `createCountNames` (บรรทัดที่ 12) ทำหน้าที่สร้างอาเรย์รายชื่อของตัวนับแต่ละตัว โดยตรวจสอบว่าได้มีการกำกับมาใน `countNames` หรือไม่ โดยที่ตัวนับแต่ละตัวจะถูกกำกับชื่อให้ตามลำดับ ดังที่แสดงใน รหัสที่ 4.7

```
private void createCountNames(ProfileValue profValue, int countNo) {
        //count name
        countNames = new String[countNo];
        if (profValue.getCountNames().length == 0) {
            if (profValue.getCountNo() == 1) {
                countNames[0] = "";
```

```
            } else {
                for (int ind = 0; ind < countNo; ind++) {
                    countNames[ind] = Integer.toString(ind);
                }
            }
        } else if (profValue.getCountNames().length == countNo) {
            String[] tempCountName = profValue.getCountNames();
            for (int ind = 0; ind < countNo; ind++) {
                if (!tempCountName[ind].equalsIgnoreCase("")) {
                    countNames[ind] = tempCountName[ind];
                } else {
                    countNames[ind] = Integer.toString(ind);
                }
            }
        } else {
            assert false;
        }
    }
```

รหัสที่ 4.7 รหัสเทียมการสร้างอาเรย์รายชื่อของตัวนับ

- `createAdjustMethods` (บรรทัดที 13) ทำหน้าที่สร้างอาเรย์เมท็อดปรับข้อมูลตามที่กำกับใน adjustFunctions (หัวข้อ 3.4.8) สำหรับการปรับค่าตัวนับ โดยที่ตัวนับแต่ละตัวจะถูกกำกับ เมท็อดปรับข้อมูลเรียงตามลำดับ การสร้างเมท็อดปรับข้อมูลแต่ละตัวอาศัย Java Reflection API ดังที่แสดงในรหัสที่ 4.8 และเมท็อดปรับข้อมูลจะต้องรับพารามิเตอร์เป็นข้อมูลชนิด `double` สองตัว และคืนค่าเป็นข้อมูลชนิด `double` เช่นกัน

```
private void createAdjustMethods(ProfileValue profValue, int countNo)
                                                        throws Exception {
        Object intstanc = null;
        if (profValue.getAdjustFunctions().length > 0) {
            if (profValue.getAdjustFunctions().length != countNo) {
                throw new Exception(" lengh of AdjustFunctions [" +
                   profValue.getCountNames().length + "]) != countNo [" +
                   countNo + "]");
            }
            adjMethods = new Method[countNo];
            for (int indCount = 0; indCount < countNo; indCount++) {
                String adjFuncName = profValue.getAdjFunctions()[indCount];
                if (!adjFuncName.equalsIgnoreCase("")) {
                    Constructor constructor = obj.getConstructor();
                    intstanc = constructor.newInstance();
                    adjMethods[indCount] = obj.getMethod(adjFuncName,
                                           double.class, double.class);
                }
            }
        }
}
```

รหัสที่ 4.8 รหัสเทียมการสร้างอาเรย์เมท็อดปรับข้อมูล

- บรรทัดที 18 เป็นวงวนหยิบหน่วยผลิตข้อมูลแต่ละแบบมาผลิตข้อมูลตามที่กำหนดใน `from`, `to`, และ `step` ในวงวนบรรทัดที 24 แล้วเข้าอีกวงวนในบรรทัดที 26 เพื่อสั่ง

เมท็อดที่สนใจทำงานซ้ำตามจำนวนครั้งที่ `repeat` แต่ละครั้งที่ทำงาน (บรรทัดที่ 30) ให้รับข้อมูลที่ผลิตจากหน่วยผลิตข้อมูลขาเข้า (บรรทัดที่ 28) หลังจากเมท็อดทำงานเสร็จ ก็ปรับค่าจำนวนครั้ง (บรรทัดที่ 33) และรวบรวมจำนวนครั้งที่ได้นับไว้ (บรรทัดที่ 34) เมื่อทำซ้ำครบก็คำนวณค่าเฉลี่ย (บรรทัดที่ 36) แล้วแจ้งให้หน่วยแสดงผลทราบ (บรรทัดที่ 38)

- การเก็บเวลาที่เมท็อดใช้ในการทำงาน เริ่มจากบันทึกเวลาก่อนเรียกให้เมท็อดทำงาน (บรรทัดที่ 29) นำไปลบออกจากเวลาที่หลังจากเมท็อดทำงานเสร็จ (บรรทัดที่ 30) แล้วเก็บเป็นรายการของเวลาที่ใช้ในแต่ละรอบ (บรรทัดที่ 31) เมื่อทำงานครบตามจำนวนครั้ง `repeat` แล้วคำนวณค่าเฉลียโดยตัดข้อมูลส่วนที่มีค่ามาก และข้อมูลส่วนที่มีค่าน้อยออก คิดเป็นร้อยละตามที่กำหนดใน `timeTruncate` (บรรทัดที่ 39) แล้วแจ้งให้หน่วยแสดงผลทราบ (บรรทัดที่ 40)

- การทำปรับเส้นโค้งผลลัพธ์เริ่มจากกำหนดหน่วยผลิตข้อมูลขาเข้า (บรรทัดที่ 22) ระหว่างทำการทดลองจะเก็บค่าเฉลี่ยการทำงานและขนาดข้อมูลขาเข้า (บรรทัดที่ 37) เมื่อครบแล้วจะคำนวณผลการทดลองที่รวบรวมมาหาคำตอบที่เหมาะสม โดยใช้วิธีการ Levenberg-Marquardt [14] อาศัยคลังคลาส LMA-Package [15] ว่ามีความสัมพันธ์ใกล้เคียงกับฟังก์ชันแบบใดที่สุด (บรรทัดที่ 44) จากแบบต่าง ๆ ดังนี้

1. $constant$
2. $\ln n$
3. $n$
4. $\mathrm{n} \ln n$
5. $n^2$
6. $n^3$

- การบันทึกข้อมูลการทดลอง เตรียมไว้สำหรับการวิเคราะห์เชิงทดลองแบบใช้ผลการทดลองเดิมเริ่มจากสร้าง `expDataMgr` (บรรทัดที่ 6) ทำหน้าที่เก็บข้อมูลการทดลองของเมท็อดที่กำกับด้วย @Profile ในการสร้างตัวเก็บข้อมูลการทดลองของเมท็อดจะเก็บชื่อเมท็อดและจำนวนตัวนับที่ใช้ไว้ด้วย เพื่อป้องกันการเปลี่ยนจำนวนตัวนับในการวิเคราะห์โดยใช้ข้อมูลเดิม (บรรทัดที่ 14) ภายในตัวเก็บข้อมูลการทดลองของเมท็อดจะแยกเก็บข้อมูลเป็นการทดลองของแต่ละหน่วยผลิตข้อมูลขาเข้า ซึ่งเก็บค่าตัวนับแต่ละตัวในการทำงานทุกรอบ และเวลาในการทำงานเฉลี่ย (บรรทัดที่ 41) และเพิ่มตัวเก็บข้อมูลผลการทดลองของเมท็อดเข้าไปที่ `expDataMgr` หลังจากทำการทดลองของเมท็อดเสร็จ (บรรทัดที่ 43) เมื่อการทำการทดลองเสร็จจึงบันทึกข้อมูลผลการทดลองเป็นแฟ้มข้อมูล (บรรทัดที่ 47) โดยอาศัยคลาส `ObjectOutputStream` ดังแสดงในรหัสที่ 4.9

```
public void writeData() {
   ObjectOutputStream outputStream = null;

   // jed : JProfile Experiment Data
   String fName = className + ".jed";
   outputStream = new ObjectOutputStream(new FileOutputStream(fName));
   outputStream.writeObject(experimantDataMethodList);
   outputStream.flush();
   outputStream.close();
}
```

รหัสที่ 4.9 รหัสเทียมการบันทึกข้อมูลการทดลอง

- `restoreSourceFile` (บรรทัดที่ 49) นำสำเนารหัสต้นฉบับที่คัดลอกไว้กลับมาแทนที รหัสต้นฉบับที่ถูกแก้ไข

การนำเสนอผลการทดลองเดิมในรูปแบบใหม่ จะช่วยให้ประหยัดเวลาในการวิเคราะห์เชิงทดลองที่ต้องการดูผลการทดลองหลาย ๆ รูปแบบ เนื่องจากการทดลองแต่ละครั้งอาจใช้เวลานาน ตัวควบคุมการทดลองจะทำงานคล้ายกับการทำงานแบบทดลองทำงานจริง เพียงแต่แทนที่จะทำงานจริงตัวควบคุมจะใช้ข้อมูลที่ถูกบันทึกไว้แล้วแทน ดังแสดงในรหัสที่ 4.10

```
01:  public class Profiler {
02:  void begin(clsName) {
03:   init();
04:   instrumentation(clsName);
05:   classProf = cls.getProfileAnnotation();
06:   expDataMgr = createExpDataMgr(className);
07:   //-------------------------
08:   for (method : cls.getProfiledMethods()) {
09:    methodProf = method.getProfileAnnotation();
10:    profValue.extractProfile(classProf, methodProf);
11:    addOutputListeners(profValue);
12:    expDataMethod = expDataMgr.getData(method.getName());
13:    countNo = expDataMethod.getCountNo();
14:    countNames  = createCountNames(profValue, countNo);
15:    adjustMethods  = createAdjustMethods(profValue, countNo);
16:    fireEvent(Event.METHOD, method.getName(), countNo, countNames);
17:    //-------------------------
18:    for (expData : expDataMethod.getDataList()) { //input
19:     fireEvent(Event.INPUT, expData.input);
20:     fitMgr.newInput(expData.ingen);
21:     //-------------------------
22:     for (x = 0; x <= expData.getXSize(); x++) { //x
23:      tempYCounts =  experimentDataInfo.getAllYs(x);
24:      for (c = 0; c < tempYCounts.length (); c++) { //countNo
25:       resetCounts();
26:       for (r = 0; r < tempYCounts.length (); r++) {//repeat
27:          tempCount =  tempYCounts[c][r];
28:          adjCount(tempCount);
29:          sumCount(tempAdjCount);
30:       }
31:       avg[c] = avgCounts(sumAdjCount);
32:      }
33:      fitMgr.newData(expData.X(n),avg);
```

```
34:      fireEvent(Event.DATA, expData.X(n), avg);
35:      time = avgTime(expData.times(n), profValue.timeTruncate());
36:      fireEvent(Event.TIME, expData.X(n), time);
37:     }
38:     expDataMgr.addData(expDataMethod)
39:     fireEvent(Event.FIT, n, fitMgr.fit());
40:    }
41:   }
42:   fireEvent(Event.FINISH, null);
43:   recoverSourceFile();
44:  }
45:…
```

รหัสที่ 4.10 รหัสเทียมของตัวควบคุมกรณีใช้ผลการทดลองเดิม

- เนื่องจากตัวควบคุมจะยังไม่ทราบว่าผู้ใช้กำหนดให้ใช้ผลการทดลองเดิมหรือไม่ และไม่สามารถนำเข้ารหัสไบต์ของคลาสที่ทำการวิเคราะห์ได้อีกในภายหลัง จึงต้องเรียกเมท็อด `instrumentation` (บรรทัดที่ 4) ด้วย ซึ่งทำงานเหมือนในการทำงานแบบทดลองจริง

- สร้าง `expDataMgr` (บรรทัดที่ 6) ซึ่งอ่านข้อมูลผลการทดลองจากแฟ้มข้อมูล โดยอาศัยคลาส `ObjectInputStream` ดังแสดงในรหัสที่ 4.11 ซึ่งจะได้ อาร์เรย์ลิสต์ของ `ExperimentDataMethod`

```
public void readData() {
    ObjectInputStream inputStream = null;
    // jed : JProfile Experiment Data
    String fName = className + ".jed";
    inputStream = new ObjectInputStream(new FileInputStream(fName));
    expDataMethodList = (ArrayList) inputStream.readObject();
}
```

รหัสที่ 4.11 รหัสเทียมการอ่านข้อมูลผลการทดลองเดิม

- ในวงวนของเมท็อดที่ถูกกำกับด้วย @Profile (บรรทัดที่ 8) จะดึงข้อมูลผลการทดลองของเมท็อดออกมา โดยอาศัยชื่อของเมท็อด (บรรทัดที่ 12) และ ดึงข้อมูลจำนวนตัวนับ (บรรทัดที่ 13) จากนั้นสร้างอาเรย์รายชื่อตัวนับ และ อาเรย์เมท็อดปรับข้อมูล

- วงวนบรรทัดที่ 18 ดึงข้อมูลผลการทดลองของหน่วยผลิตข้อมูลขาเข้าแต่ละตัวออกมา บรรทัดที่ 22 วงวนของแต่ละขนาดข้อมูลขาเข้า ดึง `tempYCounts` (บรรทัดที่ 23) เป็นอาเรย์สองมิติเก็บค่าของตัวนับ โดยกำหนดให้แนวแถวเป็นตัวนับและแนวหลักเป็นการทำงานแต่ละรอบ ซึ่งใช้วงวนบรรทัดที่ 24 และวงวนบรรทัดที่ 26 ในการอ่านค่าใน `tempYCounts` ออกมา (บรรทัดที่ 27) อาจมีการเรียกเมท็อดปรับค่าของตัวนับนั้นถ้ากำกับเมท็อดปรับค่า (บรรทัดที่ 28) รวมค่าตัวนับ (บรรทัดที่ 29) แล้วหาค่าเฉลี่ยของตัวนับแต่ตัวของขนาดข้อมูลขาเข้า (บรรทัดที่ 31)

# บทที่ 5

# ตัวอย่างและผลการทดลอง

บทนี้จะแสดงตัวอย่างการวิเคาะห์อัลกอริทึมเชิงทดลอง โดยใช้งานตัวควบคุมและคลาสอรรถประโยชน์สำหรับการวิเคราะห์อัลกอริทึมเชิงทดลอง

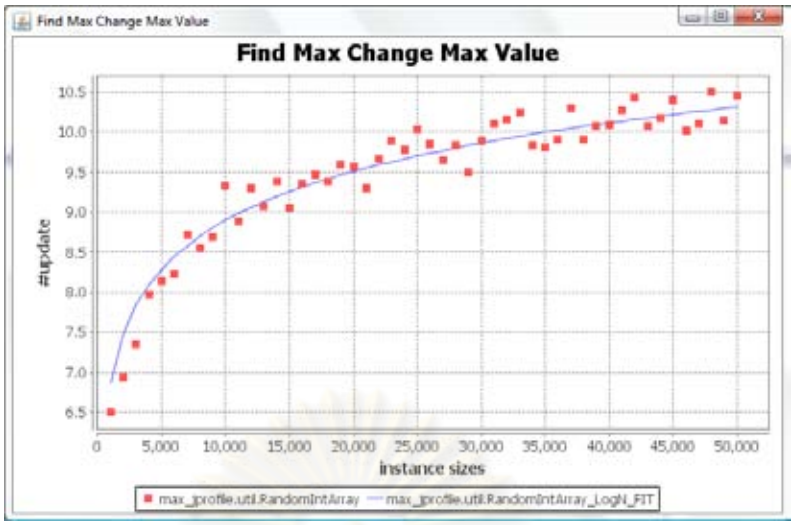
## 5.1 ตัวอย่างการวิเคราะห์การเปลี่ยนค่ามากสุดในระหว่างการหาค่ามากสุด

วิธีการหาค่ามากสุดในอาเรย์ที่เร็วและง่ายที่สุด คือ การไล่เปรียบเทียบข้อมูลในอาเรย์ทุกตัวจากตัวแรกไปถึงตัวสุดท้าย โดยจำค่าที่มากที่สุดของชุดข้อมูลที่ผ่านมาไว้ และเปรียบเทียบกับข้อมูลตัวถัดไปถ้าข้อมูลตัวถัดไปมากกว่าก็เปลี่ยนค่ามากสุดเป็นข้อมูลตัวนั้น ถ้าข้อมูลตัวแรกเป็นข้อมูลที่มีค่ามากที่สุดจะไม่มีการเปลี่ยนค่ามากที่สุดที่เก็บไว้ และถ้าข้อมูลเรียงจากน้อยไปหามากค่าที่จำไว้จะเปลี่ยนทุกรอบที่มีการเปรียบเทียบ สิ่งที่สนใจคือโดยเฉลี่ยจะมีการเปลี่ยนค่ามากสุดกี่ครั้ง

รหัสที่ 5.1 แสดงแสดงเมท็อด max ที่แสดงวิธีการหาค่ามาที่สุดในอาเรย์ data จากนั้นเขียนคำสั่ง `//@Profiler.count++` ไว้ที่หน้าคำสั่งที่สนใจในที่นี่คือคำสั่ง `max = data[i];` เพราะเป็นคำสั่งเปลี่ยนค่ามากที่สุด

```
public class FineMax {
  @Profile(name = "Find Max Change Max Value",
           ytitle="#update",
           inputs = {jprofile.util.RandomIntArray.class},
           outputs = {jprofile.output.ScatterPlot.class},
           from = 1000, to = 50000, step=1000, repeat=300,
           curveFitting=true(
  public int max(int[] data) {
    int max = data[0];
    for (int i = 1; i < data.length; i++) {
      if (max < data[i]) {
        //@Profiler.count++;
        max = data[i];
      }
    }
    return max;
  }
}
```

รหัสที่ 5.1 การทดลองหาค่าเฉลี่ยการเปลี่ยนค่ามากสุด

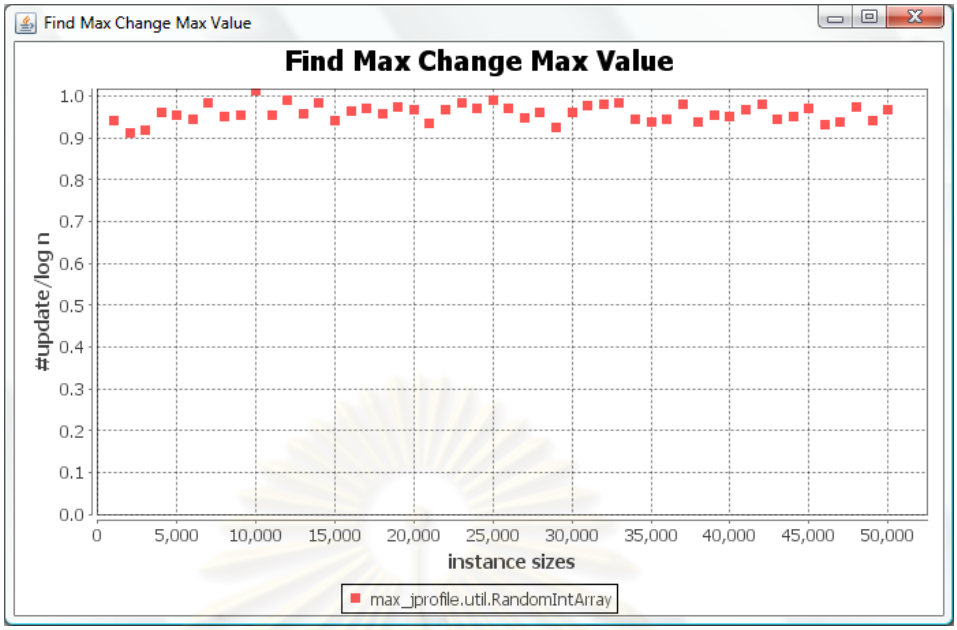



รูปที่ 5.1 ผลการทดลองหาค่าเฉลี่ยค่ามากสุด แนวโน้มแบบ log n

จากผลการทดลองในรูปที่ 5.1 จะเห็นได้ว่าจำนวนครั้งของการเปลี่ยนค่ามากสุดมีการเปลี่ยนแปลงเพิ่มตามขนาดของอาเรย์ แต่เพิ่มด้วยอัตราการเติบโตที่น้อยกว่าเชิงเส้น มองด้วยตา และดูจากการปรับเส้นโค้งผลลัพธ์ อาจสรุปคร่าว ๆ ว่ามีอัตราเติบโตแบบลอกการิทึม ของขนาดอาเรย์ ซึ่งหากต้องการยืนยันการคาดการณ์นี้ก็ต้องปรับผลการทดลองใหม่โดยใช้ผลการทดลองเดิม ให้จำนวนการเปลี่ยนค่ามากสุด หารด้วย $\ln n$ ซึ่งทำได้โดยเขียนเมท็อด divideByLog และกำหนด adjustFunctions พร้อมเปลี่ยนข้อความกำกับแกน Y ใน @Profile ดังแสดงใน รหัสที่ 5.2 เมื่อทำการปรับการแสดงผลแล้วได้ผลดังรูปที่ 5.2 แสดงให้เห็นว่า ค่าของจำนวนการเปลี่ยนตัวมากสุดหารด้วย $\ln n$ เป็นค่าคงตัว เป็นการยืนยันว่าอัตราการเปลี่ยนค่ามากสุดใน max นั้นมีอัตราการเติบโตแบบ logarithm ซึ่งสอดคล้องกับการวิเคราะห์เชิงคณิตศาสตร์ของ Knuth [4] ที่แสดงให้เห็นว่าจำนวนการเปลี่ยนค่ามากสุดโดยเฉลี่ย มีค่าเท่ากับ $H_n - 1$ โดยที่ $H_n$ คือจำนวนฮาร์มอนิกที่ n มีค่าประมาณ $\ln n$ เมื่อ n มีค่ามาก

```
public class FineMax {
  @Profile(name = "Find Max Change Max Value",
           ytitle="#update/log n",
           ...
           useOldData=true , adjustFunctions={"divideByLog"}
(
  public int max(int[] data) {
    ...
  }
  public double divideByLog(double x, double y) {
        return y / Math.log(x);
  }
}
```

รหัสที่ 5.2 การทดลองหาค่าเฉลี่ยการเปลี่ยนค่ามากสุด โดยปรับผลการทดลอง และใช้ผลการทดลองเดิม




รูปที่ 5.2 ผลที่ได้ว่า #update / log n เป็นค่าคงตัว

## 5.2 ตัวอย่างการวิเคราะห์จำนวนการเปรียบเทียบข้อมูลในการค้นข้อมูลในรายการที่ปรับตัวเองได้แบบ move-to-front

การค้นข้อมูลในรายการที่ปรับตัวเองได้แบบ move-to-front [16] เป็นการค้นหาข้อมูลในรายการแล้วย้ายข้อมูลที่พบไปไว้ตำแหน่งหน้าสุดของรายการ โดยหวังว่าจะทำให้การค้นหาครั้งถัดไปเร็วขึ้น การวิเคราะห์นี้สนใจว่าในการค้นหาข้อมูลในรายการที่ปรับตัวเองได้แบบ move-to-front แล้วหวังว่าการค้าหาข้อมูลจะเร็วขึ้นนั้นจะเร็วขึ้นขนาดไหน เทียบกับการค้นหาข้อมูลในรายการที่ไม่ปรับตัวเอง รหัสที่ 5.3 แสดงเมท็อด `search` ที่ค้นหาข้อมูลในรายการแล้วย้ายข้อมูลนั้นไปไว้ตำแหน่งแรกของรายการ กำหนดให้วิเคราะห์จำนวนครั้งการเปรียบเทียบ จึงเขียนคำสั่ง `//@Profiler.count++` ไว้หน้าคำสั่งเปรียบเทียบ ให้สังเกตุว่าเมท็อด `search` รับพารามิเตอร์ 2 ตัว คือรายการ `d` และจำนวนเต็ม `x` หน่วยผลิตข้อมูลขาเข้าจะต้องผลิตข้อมูลทั้งรายการ `d` และจำนวนเต็ม `x` ในที่นี่เราต้องการจะเปรียบเทียบระหว่างลักษณะข้อมูลขาเข้า 2 แบบ จึงเขียน annotation ให้ใช้หน่วยผลิตข้อมูล 2 แบบ คือ `RandomIntList` และ `RandomZipfIntList` แบบแรกเป็นการสร้างรายการจำนวนเต็มแบบสุ่มแล้วสุ่มหยิบข้อมูลออกมาเป็นจำนวนเต็ม `x` ดังแสดงในรหัสที่ 5.4 แบบที่สองเป็นการสร้างรายการจำนวนเต็มแบบสุ่มแต่หยิบข้อมูลออกมาเป็นจำนวนเต็ม x มีการกระจายแบบ Zipf ($P_i = 1/(iH_n)$ หมายความว่าข้อมูลที่ถูกค้นบ่อยเป็นอันดับ k จะมีโอกาสถูกค้นน้อยกว่าตัวที่ k-1 เป็นจำนวน k เท่า) ดังแสดงในรหัสที่ 5.5 ซึ่งสะท้อนสภาพการค้นหาข้อมูลที่ใกล้เคียงความเป็นจริงในทางปฏิบัติ [17]

```
@Profile(name = “Move to front", xtitle = "N", ytitle = "Count",
    inputs = {RandomIntList.class,
              RandomZipfIntList.class},
    outputs = {jprofile.output.ScatterPlot.class},
    from = 1000, to = 10000, step = 500, repeat = 300,seed=1000L,
    curveFitting = true)
public boolean search(List d, int x) {

        for (Iterator i = d.iterator(); i.hasNext();) {
            //@Profiler.count++;
            if (((Integer) i.next()) == x) {
                i.remove();
                d.add(0, x); // ย้ายข้อมูลไปต้นรายการ
                return true;
            }
        }
        return false;
}
```

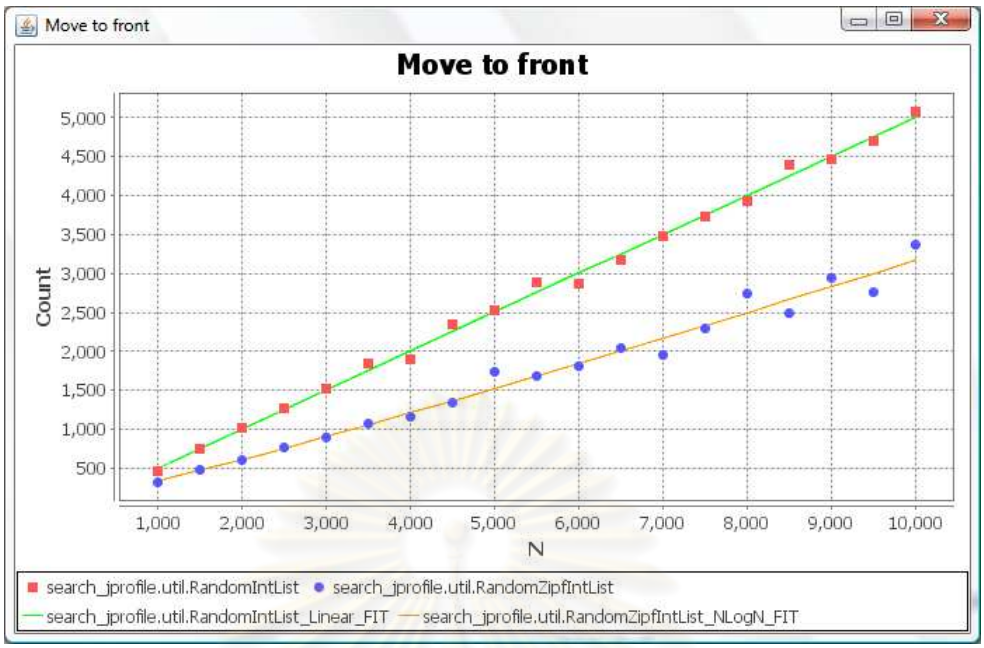
รหัสที่ 5.3 การทดลองหาจำนวนครั้งของการเปรียบเทียบเพื่อค้นข้อมูลในรายการที่ปรับตัวเองได้แบบ move-to-front

```
public class RandomIntList implements InputGenerator{
    private int[] d;
    private List dList;
    private int n;
    private RandomIntList() {
        n = 0;
    }
    public void init(int n) {
        d = new int[n];
        dList = new ArrayList();
        for (int i = 0; i < d.length; i++) { d[i] = i; }
        Random rgen = new Random();
        //--- Shuffle by exchanging each element randomly
        for (int i = 0; i < d.length; i++) {
            int randomPosition = rgen.nextInt(d.length);
            int temp = d[i];
            d[i] = d[randomPosition];
            d[randomPosition] = temp;
        }
        for (int i = 0; i < d.length; i++){ dList.add(d[i]); }
    }
    @Override
    public Object[] get(int aN) {
        if(n != aN){
            n = aN;
            init(n);
        }
        int i = (int) (n * Profiler.random.nextDouble());
            return new Object[]{dList, new Integer(i)};
    }
}
```

รหัสที่ 5.4 หน่วยผลิตข้อมูล RandomIntList

```
public class RandomZipfIntList implements InputGenerator{
    private int[] d;
    private List dList;
    private double[] zd;
    private int n;

    public RandomZipfIntList() {
        n = 0;
    }
    public void init(int n) {
        d = new int[n];
        dList = new ArrayList();
        for (int i = 0; i < d.length; i++) {
            d[i] = i;
        }
        Random rgen = new Random();
        //--- Shuffle by exchanging each element randomly
        for (int i = 0; i < d.length; i++) {
            int randomPosition = rgen.nextInt(d.length);
            int temp = d[i];
            d[i] = d[randomPosition];
            d[randomPosition] = temp;
        }
        for (int i = 0; i < d.length; i++){
            dList.add(d[i]);
        }
        double zn = 0;
        for (int i = 1; i <= n; i++) {
            zn += 1.0 / i;
        }
        zd = new double[n];
        for (int i = 0; i < n; i++) {
            zd[i] = 1.0 / (i + 1) / zn;
        }
        for (int i = 1; i < n; i++) {
            zd[i] += zd[i - 1];
        }
    }
    @Override
    public Object[] get(int aN) {
        if(n != aN){
            n = aN;
            init(n);
        }
        int i;
        double x = Profiler.random.nextDouble();
        for (i = 0; i < zd.length - 1; i++) {
            if (x < zd[i]) {
                break;
            }
        }
        return new Object[]{dList, new Integer(i)};
    }
}
```

รหัสที่ 5.5 หน่วยผลิตข้อมูล RandomZipfIntList




รูปที่ 5.3 จำนวนครั้งของการเปรียบเทียบเพื่อค้นข้อมูลด้วยการกระจายแบบสุ่ม และแบบ Zipf

การค้นหาข้อมูลในรายการที่ไม่ปรับตัวเอง โดยสุ่มข้อมูลในรายการมาค้นจะต้องเปรียบเทียบข้อมูลโดยเฉลี่ยประมาณครึ่งหนึ่งของขนาดรายการ รูปที่ 5.3 แสดงผลการทดลองการค้นหาข้อมูลในรายการที่ปรับตัวเองแบบ move-to-front เห็นได้ว่าถ้าหยิบข้อมูลแบบสุ่มมาค้นหาในรายการ การปรับตัวเองแบบ move-to-front ไม่มีผลทำให้การค้นหาข้อมูลเร็วขึ้น แต่ถ้าการหยิบข้อมูลมีการกระจายแบบ Zipf จะมีผลให้การค้นหาเร็วขึ้น Revest [18] ได้วิเคราะห์ทางคณิตศาสตร์ไว้ว่าจำนวนการเปรียบเทียบเฉลี่ยมีค่าประมาณ $1.386\,n\,/\,H_n$ จากการปรับเส้นโค้งผลลัพท์ซึ่งแสดงในส่วนคำอธิบายสัญลักษณ์ จะเห็นได้ว่ากราฟจำนวนครั้งของการเปรียบเทียบเพื่อค้นข้อมูลด้วยการกระจายแบบ Zipf มีการเติบโตแบบ $n \log n$ ซึ่งยืนยันผลการวิเคราะห์ของ Rivest

## 5.3 ตัวอย่างวิเคราะห์เพื่อเปรียบเทียบจำนวนครั้งของการเปรียบเทียบข้อมูลในการสร้างฮีปทวิภาค

การสร้างฮีปทวิภาคกระทำได้โดยค่อย ๆ ปรับข้อมูลในอาเรย์เสมือนการสร้างฮีปเล็ก ๆ ย่อย ๆ แล้วนำมาประกอบเป็นฮีปใหญ่ขึ้น ๆ หรือใช้วิธีการนำข้อมูลในอาเรย์ค่อย ๆ เพิ่มใส่ฮีป วิธีแรกใช้เวลาเป็น $O(n)$ ในขณะที่วิธีหลังใช้เวลาเป็น $O(n \log n)$ [19] รหัสที่ 5.5 แสดงเมท็อด `buildHeap1` สร้างฮีปด้วยวิธีการค่อย ๆ ปรับ และ `buildHeap2` สร้างฮีปด้วยวิธีการค่อย ๆ เพิ่ม เราต้องการวิเคราะห์เชิงทดลองโดยใช้หน่วยผลิตข้อมูล 3 แบบที่ผลิตข้อมูลเรียงลำดับจากน้อยไปมาก (`SortedIntArray`) แบบที่ผลิตข้อมูลสุ่ม (`RandomIntArray`) และแบบที่ผลิตข้อมูลเรียงกลับ

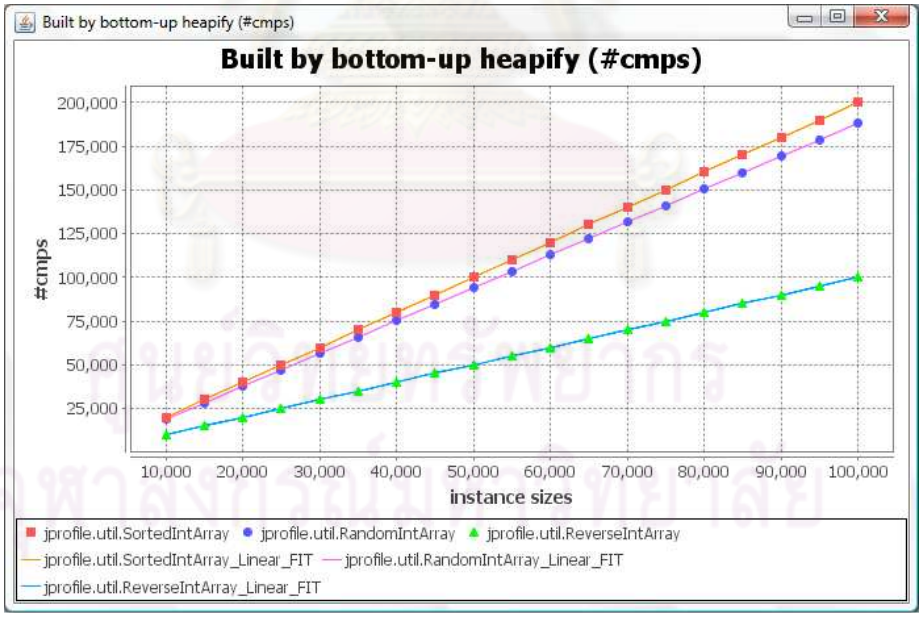
ลำดับจากมากไปน้อย (ReverseIntArray) เนื่องจากต้องการวิเคราะห์เชิงทดลองทั้งสองเมท็อดด้วยรูปแบบข้อมูลในลักษณะเดียวกัน ดังนั้นจึงเขียน @Profile ไว้ที่ระดับคลาส พร้อมกำหนดค่า seed ให้กับตัวผลิตจำนวนสุ่มของระบบด้วยเพื่อให้หน่วยผลิตข้อมูลขาเข้าผลิตชุดข้อมูลสุ่มที่เหมือนกันให้กับทั้งสองการทดลอง ที่ @Profile ระดับเมท็อดกำหนดชื่อการทดลองและกำหนด outputs ของแต่ละเมท็อดเพื่อให้แยกแสดงผลกัน อนึ่งฮีปที่สร้างนี้เป็นฮีปแบบมากสุด (max heap)

รูปที่ 5.4 แสดงให้เห็นว่าการสร้างฮีปด้วยการค่อย ๆ ปรับนั้นใช้จำนวนการเปรียบเทียบเป็น $O(n)$ สำหรับข้อมูลขาเข้าทั้งสามรูปแบบที่ทดลอง ขณะที่รูปที่ 5.5 แสดงให้เห็นว่าการสร้างฮีปด้วยการค่อย ๆ เพิ่มนั้นมีอัตราการเติบโตของจำนวนการเปรียบเทียบต่างกัน โดยจะเป็น $O(n)$ เมื่อข้อมูลขาเข้าเป็นแบบสุ่มและแบบเรียงจากน้อยไปหามาก แต่ถ้าข้อมูลขาเข้าเป็นแบบเรียงจากมากไปหาน้อยอัตราการเติบโตของจำนวนการเปรียบเทียบนั้นจะเป็น $O(n \log n)$ ซึ่งสอดคล้องกับการวิเคราะห์ทางคณิตศาสตร์ แต่ถ้าเป็นกรณีเฉลี่ยคือข้อมูลแบบสุ่ม จำนวนการเปรียบเทียบมีอัตราเติบโตเป็น $O(n)$

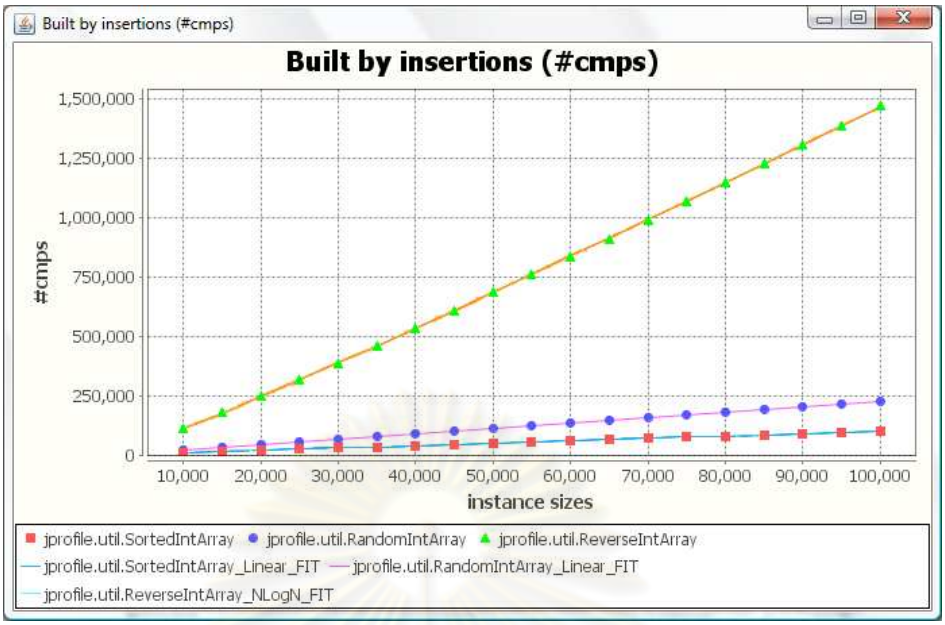
```
@Profile(seed = 1000, ytitle = "#cmps",
         inputs = {jprofile.util.SortedIntArray.class,
                   jprofile.util.RandomIntArray.class,
                   jprofile.util.ReverseIntArray.class},
         from = 10000, to = 100000, step = 5000, repeat = 10,
         curveFitting = true)
public class BuildHeap {
    @Profile(name = "Built by bottom-up heapify (#cmps)",
             outputs = {jprofile.output.ScatterPlot.class})
    public void buildHeap1(int[] d) {
           int size = d.length;
           for (int k = size / 2 - 1; k >= 0; k--) {
                fixDown(d, size, k);              // ค่อย ๆ ปรับ
           }
    }
    @Profile(name = "Built by insertions (#cmps)",
             outputs = {jprofile.output.ScatterPlot.class})
    public void buildHeap2(int[] d) {
        int size = d.length;
        for (int k = 1; k < d.length; k++) {
            fixUp(d, k);                        // ค่อย ๆ เพิ่ม
        }
    }
    void fixUp(int[] data, int k) {
        while (k > 0) {
            int p = (k - 1) / 2;
            //@Profiler.count++;
            if (data[k] >= data[p]) {
                break;
            }
            swap(data, k, p);
            k = p;
```

```
        }
    }
void fixDown(int[] data, int size, int k) {
        int c;
        while ((c = 2 * k + 1) < size) {
            //เปรียบเทียบ 2 ครั้ง  1) data[c] < data[c + 1] 2) data[k] >= data[c]
            //@Profiler.count += 2;
            if (c < size - 1 && (data[c] < data[c + 1])) {
                c++;
            }
            if (data[k] >= data[c]) {
                break;
            }
            swap(data, c, k);
            k = c;
        }
    }
    void swap(int[] data, int c, int k) {
        int temp = data[c];
        data[c] = data[k];
        data[k] = temp;
    }
}
```

รหัสที่ 5.6 การทดลองเปรียบเทียบจำนวนครั้งของการเปรียบเทียบเพื่อสร้างฮีปทวิภาคด้วยวิธีการค่อย ๆ เพิ่มกับวิธีการค่อย ๆ ปรับ




รูปที่ 5.4 จำนวนการเปรียบเทียบของการสร้างฮีป ด้วยการค่อย ๆ ปรับข้อมูลในอาเรย์




รูปที่ 5.5 จำนวนการเปรียบเทียบของการสร้างฮีป ด้วยการค่อย ๆ เพิ่มข้อมูล

## 5.4 ตัวอย่างวิเคราะห์เพื่อเปรียบเทียบจำนวนครั้งของการเปรียบเทียบข้อมูลและการย้ายข้อมูลในการเรียงข้อมูลโดยใช้วิธี quicksort และ mergesort

ในการวิเคราะห์อัลกอริทึมทางคณิตศาสตร์ของการเรียงข้อมูลวิธี quicksort และ mergesort ใช้เวลาเป็น $O(n \log n)$ เราต้องการศึกษาทั้งจำนวนการเปรียบเทียบและการสลับข้อมูล จึงต้องกำหนดชื่อตัวนับ 2 ตัวด้วย `@Profile countNames = {“CMP”,”MOV”}` ตัวนับตัวที่หนึ่งสำหรับการเปรียบเทียบข้อมูล (CMP) และตัวที่สองสำหรับการสลับข้อมูล (MOV) โดยไม่ต้องกำหนด countNo ซึ่งแสดงไว้ใน รหัสที่ 5.6

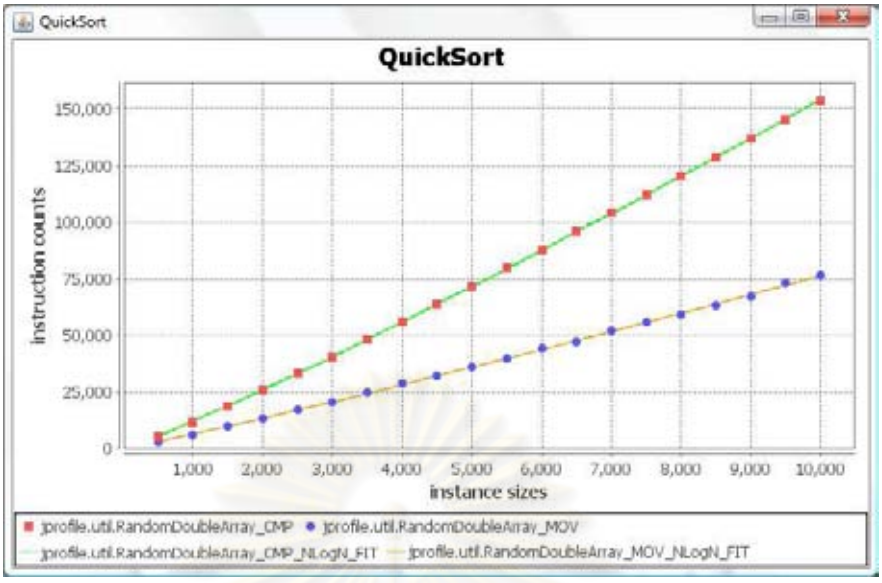
```
...
@Profile(seed=1000L,
         inputs = {RandomDoubleArray.class},
         from = 500, to = 10000, step = 500, repeat = 100,
         countNames = {"CMP", "MOV"}, curveFitting = true)
public class Sort2 {

@Profile(name = "QuickSort",
         outputs = {jprofile.output.ScatterPlot.class})
    public static void quicksort(double[] a) {
        quicksort(a, 0, a.length - 1);
    }

    public static void quicksort(double[] a, int left, int right) {
        if (right <= left) return;
        int i = partition(a, left, right);
        quicksort(a, left, i-1);
        quicksort(a, i+1, right);
    }
```

```
private static int partition(double[] a, int left, int right) {
        int i = left - 1;
        int j = right;
        while (true) {
            while (less(a[++i], a[right]))
            while (less(a[right], a[--j]))
                if (j == left) break;
            if (i >= j) break;
            exch(a, i, j);
        }
        exch(a, i, right);
        return i;
    }

    // is x < y ?
    private static boolean less(double x, double y) {
        //@Profiler.counts[0]++;
        return (x < y);
    }

    // exchange a[i] and a[j]
    private static void exch(double[] a, int i, int j) {
        //@Profiler.counts[1]+=3;  // การ swap ทำงาน 3 คำสั่งต่อ 1 ครั้ง
        double swap = a[i];
        a[i] = a[j];
        a[j] = swap;
    }

    @Profile(name="MergeSort",
             outputs = {jprofile.output.ScatterPlot.class})
    public static void mergeSort( double [ ] a ) {
        double [ ] tmpArray = new double[ a.length ];
        mergeSort( a, tmpArray, 0, a.length - 1 );
    }

    private static void mergeSort( double [ ] a, double [ ] tmpArray,
            int left, int right ) {

        if( left < right ) {
            int center = ( left + right ) / 2;
            mergeSort( a, tmpArray, left, center );
            mergeSort( a, tmpArray, center + 1, right );
            merge( a, tmpArray, left, center + 1, right );
        }
    }
```

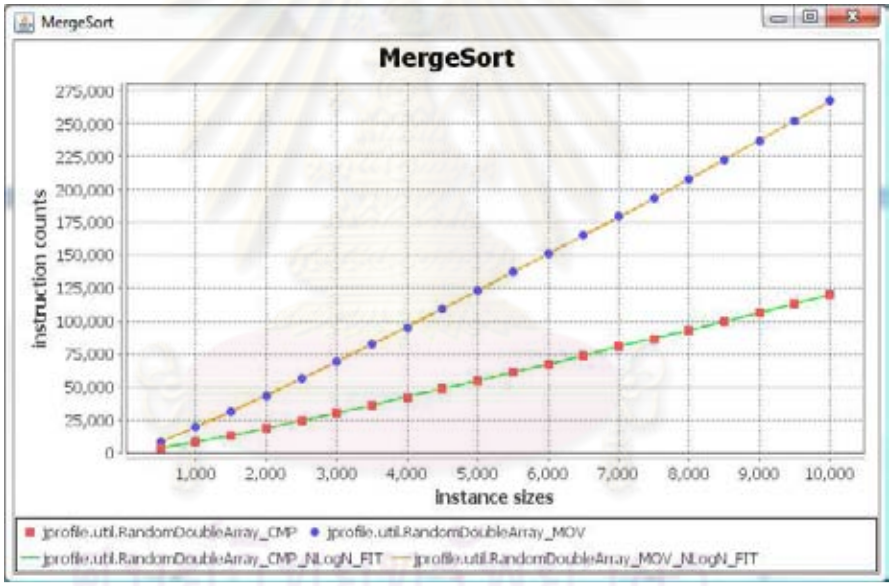
```
private static void merge( double [ ] a, double [ ] tmpArray,
            int leftPos, int rightPos, int rightEnd ) {
        int leftEnd = rightPos - 1;
        int tmpPos = leftPos;
        int numElements = rightEnd - leftPos + 1;

        // Main loop
        while( leftPos <= leftEnd && rightPos <= rightEnd ){
            //@Profiler.counts[0]++;
            //@Profiler.counts[1]++;
            if( a[ leftPos ] <=  a[ rightPos ]  )
                tmpArray[ tmpPos++ ] = a[ leftPos++ ];
            else
                tmpArray[ tmpPos++ ] = a[ rightPos++ ];
        }

        while( leftPos <= leftEnd ){    // Copy rest of first half
            //@Profiler.counts[1]++;
            tmpArray[ tmpPos++ ] = a[ leftPos++ ];
        }

        while( rightPos <= rightEnd ){  // Copy rest of right half
            //@Profiler.counts[1]++;
            tmpArray[ tmpPos++ ] = a[ rightPos++ ];
        }

        // Copy tmpArray back
        for( int i = 0; i < numElements; i++, rightEnd-- ){
            //@Profiler.counts[1]++;
            a[ rightEnd ] = tmpArray[ rightEnd ];
        }
    }
}
```

รหัสที่ 5.7 การทดลองเปรียบเทียบจำนวนครั้งของการเปรียบเทียบและการสลับข้อมูลในการเรียงข้อมูลโดยใช้วิธี quicksort และ mergesort

รูปที่ 5.6 และรูปที่ 5.7 แสดงให้เห็นว่าในการเรียงข้อมูลทั้งแบบ quicksort และแบบ mergesort จำนวนครั้งของการเปรียบเทียบและการสลับข้อมูลมีการเติบโตแบบ $O(n \log n)$ โดยแบบ quicksort จำนวนการครั้งของการเปรียบเทียบข้อมูลมากกว่าจำนวนครั้งของการสลับข้อมูล ในขณะที่แบบ mergesort จำนวนครั้งของการสลับข้อมูลมากกว่าจำนวนครั้งการเปรียบเทียบข้อมูล




รูปที่ 5.6 จำนวนการเปรียบเทียบและการย้ายข้อมูลในการเรียงข้อมูลแบบ quicksort




รูปที่ 5.7 จำนวนการเปรียบเทียบและการสลับข้อมูลในการเรียงข้อมูลแบบ mergesort

## 5.5 ตัวอย่างวิเคราะห์จำนวนครั้งของการเปรียบเทียบข้อมูลและการย้ายข้อมูลในการเรียงข้อมูลโดยใช้วิธี mergesort แบบสร้างอาเรย์ชั่วคราว กับ แบบสลับในอาเรย์เดิม

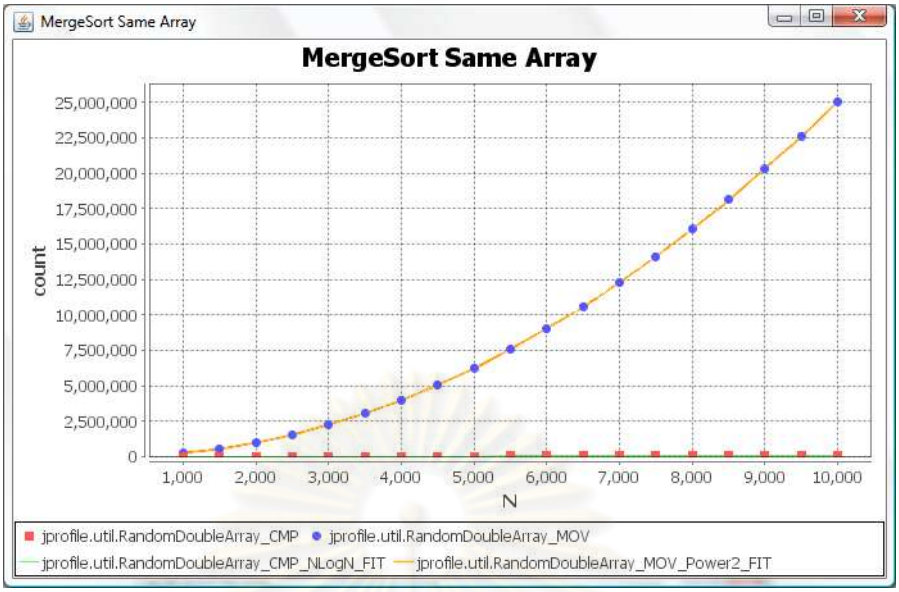
วิธีการเรียงข้อมูลแบบ mergesort คือการแบ่งข้อมูลเป็นส่วนย่อย ๆ แล้วนำข้อมูลย่อยแต่ละตัวที่เรียงลำดับแล้วมารวมเข้าด้วยกันเช่นเดิม ก็จะได้ข้อมูลที่เรียงลำดับตามต้องการ ในที่นี้สนใจวิเคราะห์จำนวนครั้งการเปรียบเทียบและการสลับข้อมูลในวิธีการรวมข้อมูลกลับ ระหว่างแบบสลับข้อมูลในอาเรย์เดิม [20] กับสร้างอาเรย์ชั่วคราว ว่าจะมีความต่างกันอย่างไร ดังแสดงใน รหัสที่ 5.7

```
@Profile(seed = 1000L,
         inputs = {RandomDoubleArray.class},
         xtitle = "N", ytitle = "count",
         from = 1000, to = 10000, step = 500, repeat = 100,
         countNames = {"CMP", "MOV"}, curveFitting = true)
public class MergeSort {

   @Profile(name="MergeSort Same Array",
            outputs = {jprofile.output.ScatterPlot.class})
    public static void mergeSortSameArray(double[] array){
          mergeSortSameArray(array,0,array.length - 1);
    }

    public static void mergeSortSameArray(double[] array, int lo, int n) {
        int low = lo;
        int high = n;
        if (low >= high) {
          return;
        }
        int middle = (low + high) / 2;
        mergeSortSameArray(array, low, middle);
        mergeSortSameArray(array, middle + 1, high);
        int end_low = middle;
        int start_high = middle + 1;
        while ((lo <= end_low) && (start_high <= high)) {
            //@Profiler.counts[0]++;
            if (array[low] < array[start_high]) {
                low++;
            } else {
                double Temp = array[start_high];
                for (int k = start_high - 1; k >= low; k--) {
                    //@Profiler.counts[1]++;
                    array[k + 1] = array[k];
                }
                //@Profiler.counts[1]++;
                array[low] = Temp;
                low++;
                end_low++;
                start_high++;
            }
        }
    }
    @Profile(name="MergeSort New Array",
             outputs = {jprofile.output.ScatterPlot.class})
    public static void mergeSortnewArray( double [ ] a ) {
        double [ ] tmpArray = new double[ a.length ];
        mergeSortnewArray( a, tmpArray, 0, a.length - 1 );
    }
```

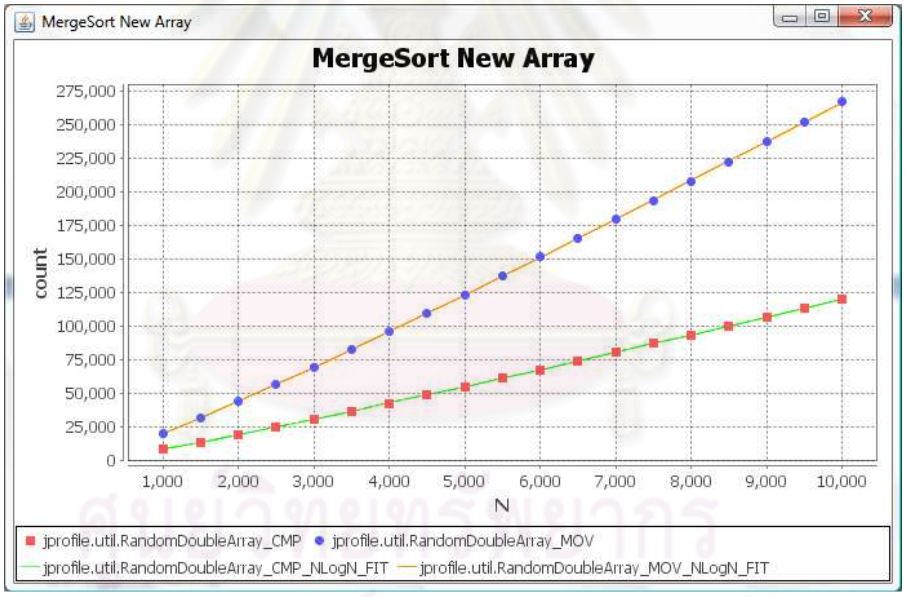
```
    private static void mergeSortnewArray( double [ ] a, double [ ]tmpArray,
            int left, int right ) {
        if( left < right ) {
            int center = ( left + right ) / 2;
            mergeSortnewArray( a, tmpArray, left, center );
            mergeSortnewArray( a, tmpArray, center + 1, right );
            merge( a, tmpArray, left, center + 1, right );
        }
    }
    private static void merge( double [ ] a, double [ ] tmpArray,
            int leftPos, int rightPos, int rightEnd ) {
        int leftEnd = rightPos - 1;
        int tmpPos = leftPos;
        int numElements = rightEnd - leftPos + 1;
        // Main loop
        while( leftPos <= leftEnd && rightPos <= rightEnd ){
            //@Profiler.counts[0]++;
            //@Profiler.counts[1]++;
            if( a[ leftPos ] <=  a[ rightPos ]  )
                tmpArray[ tmpPos++ ] = a[ leftPos++ ];
            else
                tmpArray[ tmpPos++ ] = a[ rightPos++ ];
        }

        while( leftPos <= leftEnd ){    // Copy rest of first half
            //@Profiler.counts[1]++;
            tmpArray[ tmpPos++ ] = a[ leftPos++ ];
        }
        while( rightPos <= rightEnd ){  // Copy rest of right half
            //@Profiler.counts[1]++;
            tmpArray[ tmpPos++ ] = a[ rightPos++ ];
        }
        // Copy tmpArray back
        for( int i = 0; i < numElements; i++, rightEnd-- ){
            //@Profiler.counts[1]++;
            a[ rightEnd ] = tmpArray[ rightEnd ];
        }
    }
}
```

รหัสที่ 5.8 การทดลองเปรียบเทียบจำนวนครั้งของการเปรียบเทียบและสลับข้อมูล ในการเรียงข้อมูลด้วยวิธี mergesort

รูปที่ 5.8 แสดงให้เห็นว่าการเรียงข้อมูลวิธี `mergesort` แบบสร้างอาเรย์ชั่วคราว จำนวนครั้งของการเปรียบเทียบและการสลับข้อมูลมีการเติบโตแบบ $O(n \log n)$ ส่วนเวลาในการทำงาน มีทิศทางเดียวกับจำนวนครั้งการสลับข้อมูล สำหรับแบบสลับข้อมูลในอาเรย์เดิมในรูปที่ 5.9 นั้น จำนวนครั้งการเปรียบเทียบมีการเติบโตแบบ $O(n \log n)$ เช่นกัน แต่จำนวนครั้งการสลับข้อมูลกลับมีการเติบโตแบบ $O(n^2)$ ซึ่งมีสาเหตุมาจากวิธีการสลับข้อมูลในอาเรย์เดิมจะต้องค่อย ๆ ขยับข้อมูลในส่วนแรกเพื่อนำข้อมูลในส่วนที่สองมาแทรก ทำให้ภาระงานไปอยู่ที่การย้ายข้อมูลแทนที่จะอยู่ที่การเปรียบเทียบข้อมูล สรุปได้ว่าในขั้นตอนรวมข้อมูลควรจะใช้วิธีสร้างชั่วคราว




รูปที่ 5.8 จำนวนการเปรียบเทียบ และการสลับข้อมูลในการเรียงข้อมูลแบบ mergesort แบบสลับข้อมูลในอาเรย์เดิม




รูปที่ 5.9 จำนวนการเปรียบเทียบ และการสลับข้อมูลในการเรียงข้อมูลแบบ mergesort แบบสร้างอาเรย์ชั่วคราว

# บทที่ 6
# สรุปผลการวิจัยและข้อเสนอแนะ

## 6.1 สรุปผลการวิจัย

ผู้วิจัยได้พัฒนาคลาสอรรถประโยชน์และตัวควบคุมการวิเคราะห์อัลกอริทึมเชิงทดลอง คลาสนี้ช่วยลดภาระในการทำการทดลอง เพียงผู้วิเคราะห์เขียนโปรแกรมโดยภาษาจาวากำหนดรูปแบบการทดลองและรูปแบบการแสดงผลการทดลองใน annotation @Profile จากนั้นแทรกตัวนับซึ่งสามารถมีได้หลายตัว เข้าไปยังตำแหน่งที่ต้องการเพื่อนับการทำงาน เมื่อสั่งให้ตัวควบคุมทำงาน แล้วแสดงผลการทดลอง ทำให้เห็นอัตราเติบโตของอัลกอริทึมสัมพันธ์กับขนาดของปัญหา นอกจากนี้อาจจะยังพบพฤติกรรมของอัลกอริทึมบางอย่างซึ่งการวิเคราะห์อัลกอริทึมเชิงคณิตศาสตร์ไม่ได้แสดงให้ดู และตัวควบคุมยังบันทึกผลการทดลองไว้ ผู้วิเคราะห์สามารถเรียกผลการทดลองนั้นมาแสดงผลได้อีกครั้ง ซึ่งอาจมีการปรับรูปแบบการแสดงผลไปอีก ยิ่งจะทำให้ผู้วิเคราะห์มีความเข้าใจพฤติกรรมของอัลกอริทึมมากขึ้น โดยใช้เวลาในการทดลองน้อยลง

การแทรกตัวนับเข้าไปที่รหัสต้นฉบับเพื่อแปลเป็นคลาสใหม่ในการทดลอง เป็นวิธีที่มีความซับซ้อนน้อยกว่าการแทรกคำสั่งนับในรหัสไบต์ของคลาส สามารถนำไปใช้กับ JVM ทุกตัว อีกทั้งยังทำงานได้เร็วเต็มที่ที่ JVM ทำงานได้ ทำให้สามารถทำการทดลองกับปริมาณข้อมูลขนาดใหญ่ได้ ดังแสดงในตารางที่ 6.1 จะเห็นได้ว่าเวลาในการทดลองต่างกันไม่มาก

| อัลกอริทึม | ปริมาณข้อมูลขาเข้า | | | | จำนวนการทดลอง | เวลาการทดลอง (s.) | |
|---|---|---|---|---|---|---|---|
| | จาก | ถึง | เพิ่ม | วนซ้ำ | | ไม่ใช้ JProfile101 | ใช้ JProfile101 |
| Find Max | 10,000 | 100,000 | 5,000 | 100 | 100 | 8.79 | 11.12 |
| Move to Front | 10,000 | 100,000 | 5,000 | 100 | 100 | 4.69 | 5.92 |
| Build Heap (bottom-up) | 10,000 | 100,000 | 5,000 | 100 | 100 | 10.56 | 11.63 |

ตารางที่ 6.1 เปรียบเทียบเวลาการทดลองเมื่อไม่ใช้ JProfile101 กับใช้ JProfile101

ระบบ JProfile101 ถูกพัฒนาให้มีขนาดเล็กเพียง 135 กิโลไบต์ มีความสามารถเพียงพอกับการวิเคราะห์อัลกอริทึมเชิงทดลองในวิชาการเขียนโปรแกรม วิชาโครงสร้างข้อมูล และวิชาอัลกอริทึม ในวิชาการเขียนโปรแกรมเบื้องต้นจะทำให้ตระหนักถึงความสำคัญของประสิทธิภาพเชิงเวลา โดยไม่ต้องนำเสนอรายละเอียดของการวิเคราะห์เชิงคณิตศาสตร์ และในวิชาโครงสร้างข้อมูลและอัลกอริทึมเพื่อตอกย้ำความถูกต้องผลการวิเคราะห์เชิงคณิตศาสตร์

## 6.2 ข้อเสนอแนะ

1. ระบบ JProfile101 รองรับโปรแกรมที่เขียนด้วยภาษาจาวาเท่านั้น ผู้ใช้อาจต้องการนำแนวคิดนี้ไปใช้กับภาษาอื่น ต้องพัฒนาระบบที่รองรับกับภาษานั้น
2. พัฒนาหน่วยแสดงผลให้สามารถแสดงการเปรียบเทียบระหว่างผลการทดลอง เช่น อัตราส่วน, ผลต่าง
3. พัฒนาเพิ่มการตรวจสอบหน่วยความจำที่โปรแกรมใช้ระหว่างทำการทดลอง
4. ระบบ JProfile101 ใช้วิธีการนับคำสั่งที่ผู้ใช้กำหนดให้เป็นคำสั่งตัวแทน แต่ถ้าต้องการทดลองโดยนับทุกคำสั่ง อาจใช้วิธีการกำหนดการทดลองโดยอาศัย annotation พัฒนาให้ตัวควบคุมการทดลองแบบทำการนับการทำงานทุกคำสั่ง
5. ระบบ JProfile101 รองรับโปรแกรมที่ขนาดของปัญหาขึ้นกับ 1 ตัวแปรเท่านั้น เช่น ขนาดของอาเรย์ แต่มีบางปัญหาที่ขนาดของปัญหาขึ้นกับตัวแปรมากกว่า 1 ตัวแปร เช่น จำนวนปม และ จำนวนเส้น

# รายการอ้างอิง


[1] Sedgewick, R., and Flajolet, P. An Introduction to the Analysis of Algorithm. Massachusetts: Addison-Wesley Professional, 1995.

[2] Johnson, D. S. A Theoretician's Guide to the Experiment Analysis of Algorithm, in Goldwasser, M. H., Johnson, D. S., and McGeoch, C. C. (ed.), Data Structures, Near Neighbor Searches, and Methodology: Fifth and Sixth DIMACS Implementation Challenges, pp.215-250. Providence, RI: American Mathematical Society, 2002.

[3] Sun MicroSystems, Inc. Anotation [Online]. 2007. Available from : http://java.sun.com/javase/6/docs/technotes/guides/language/annotations.html [2008, Jul 9]

[4] Knuth, D. The Art of Computer Programming, Addison-Wesley Professional, 1969

[5] Sun MicroSystems, Inc. Java Virtual Machine Profiler Interface (JVMPI) [Online]. 2007. Available from : http://java.sun.com/j2se/1.5.0/docs/guide/jvmpi.html [2008, Jul 9]

[6] Sun MicroSystems, Inc. JVM Tool Interface [Online]. 2007. Available from : http://java.sun.com/j2se/1.5.0/docs/guide/jvmti/jvmti.html [2008, Jul 9]

[7] Sun MicroSystems, Inc. HPROF: A Heap/CPU Profiling Tool in J2SE 5 [Online]. 2004. Available from : http://java.sun.com/developer/technicalArticles/Programming/HPROF.html [2008, Jul 8]

[8] Sedlacek, J. and Hurka, T. Virsual VM [Online]. (n.d.). Available from : http://visualvm.dev.java.net [2008, Jul 23]

[9] NetBeans.org. NetBeans Profiler [Online]. (n.d.). Available from : http://profiler.netbeans.org [2008, Jul 8]

[10] Binder, W. and Hulaas, J. Exact and Portable Profiling for the JVM Using Bytecode Instruction Counting. Electronic Notes in Theoretical Computer Science 164, 3 (Oct 2006): 45-64.

[11] Kuperberg, M. Krogmann, M., and Reussner, R. ByCounter: Portable Runtime Counting of Bytecode Instructions and Method Invocations. In Proceedings of ETAPS 2008, 11th European Joint Conferences on Theory and Practice of Software, 2008.

[12] Sun MicroSystems, Inc. The Java Programming Language Compiler [Online]. 2008. Available from : http://java.sun.com/javase/6/docs/technotes/guides/javac/ [2008, Jul 8]

[13] Sun MicroSystems, Inc. The Java Reflection API [Online]. 2008. Available from : http://java.sun.com/javase/6/docs/technotes/guides/reflection/index.html [2008, Jul 8]

[14] Ranganathan, A. The Levenberg-Marquardt Algorithm [Online]. 2004. Available from : http://www.cc.qatech.edu/~ananth/docs/lmtut.pdf [2009, Jan 28]

[15] Holopainen, J. LMA-Package [Online]. 2007. Available from : http://virtualrisk.cvs.sourceforge.net/*checkout*/virtualrisk/util/lma/lma_v1.3.zip [2009, Apr 23]

[16] Hester, J., and Hirschberg, D. Self-organizing linear search. ACM Computing Survey 17, 3 (Sep 1985): 295-311.

[17] Li, W. Random Texts Exhibit Zipf's Law Like Word Frequency Distribution, IEEE Transaction on Information Theory 38, 6 (Nov 1992): 1842 - 1845.

[18] Rivest, R. On self-organizing sequential search heuristic, Communication of the ACM 19, 2 (Feb 1976): 63-67.

[19] Weiss, M. Data Structures & Algorithm Analysis in Java. Boston, MA: Addison Wesley, 1998.

[20] Rose India. Merge Sort in Java [Online]. Available from : http://www.roseindia.net/java/beginners/arrayexamples/mergeSort.shtml [2009, Jul 21]

[21] Gilbert, D. JFreeChart [Online]. 2005. Available from : http://www.jfree.org/jfreechart/ [2008, Sep 20]

[22] The Apache Software Foundation. POI-HSSF – Java API To Access Microsoft Excel Format Files [Online]. 2002. Available from : http://poi.apache.org/hssf/index.html [2008, Aug 29]

# ภาคผนวก

# ภาคผนวก ก รายละเอียด annotation @Profile

ก.1 รหัสคำสั่ง annotation `@Profile`

```
import java.lang.annotation.*;
@Retention(RetentionPolicy.RUNTIME)
public @interface Profile {
    String name() default "";
    String xtitle() default "instance sizes";
    String ytitle() default "instruction counts";
    long seed() default 0;
    Class[] outputs() default {};
    Class[] inputs() default {};
    int from() default 10;
    int to() default 100;
    int step() default 1;
    int repeat() default 1;
    int countNo() default 1;
    double maxY() default Double.MAX_VALUE;
    String[] countNames() default {};
    boolean displayAvg() default true;
    boolean displayMin() default false;
    boolean displayMax() default false;
    boolean displayTime() default false;
    double timeTruncate() default 0;
    boolean curveFitting() default false;
    boolean useOldData() default false;
    String[] adjustFunctions() default {};
}
```

รหัสที่ ก.1 รายละเอียด annotation @Profile

# ภาคผนวก ข ตัวอย่างรหัสต้นฉบับหน่วยผลิตข้อมูลขาเข้าและหน่วยแสดงผล

ข.1 รหัสคำสั่งหน่วยผลิตข้อมูลขาเข้า `RandomIntArray` ผลิตข้อมูลในอาเรย์แบบจำนวนเต็มแบบสุ่ม

```
public class RandomIntArray implements InputGenerator{
    public Object[] get(int n) {
        int[] d = new int[n];
        for(int i = 0 ; i < d.length ; i++){
            d[i] = (int) (100000000*Math.random());
        }
        return new Object[]{d};
    }
```

รหัสที่ ข.1 หน่วยผลิตข้อมูลขาเข้า RandomIntArray

ข.2 รหัสคำสั่งหน่วยผลิตข้อมูลขาเข้า `SortedIntArray` ผลิตข้อมูลในอาเรย์แบบจำนวนเต็มแบบเรียงจากน้อยไปหามาก

```
public class SortedIntArray implements InputGenerator{
    public Object[] get(int n) {
        int[] d = new int[n];
        for(int i = 0 ; i < d.length ; i++){
            d[i] = i;
        }
        return new Object[]{d};
    }
}
```

รหัสที่ ข.2 หน่วยผลิตข้อมูลขาเข้า SortedIntArray

ข.3 รหัสคำสั่งหน่วยผลิตข้อมูลขาเข้า `ReverseIntArray` ผลิตข้อมูลในอาเรย์แบบจำนวนเต็มแบบเรียงจากมากไปหาน้อย

```
public class ReverseIntArray implements InputGenerator{
    public Object[] get(int n) {
        int[] d = new int[n];
        for(int i = 0 ; i < d.length ; i++){
            d[i] = n - i;
        }
        return new Object[]{d};
    }
}
```

รหัสที่ ข.3 หน่วยผลิตข้อมูลขาเข้า ReverseIntArray

ข.4 รหัสคำสั่งหน่วยผลิตข้อมูลขาเข้า `RandomIntList` ผลิตข้อมูลในรายกายจำนวนเต็มแบบสุ่ม พร้อมสุ่มเลือกข้อมูลออกมาเป็นจำนวนเต็ม `x` เมื่อเรียกเมท็อด `get` ครั้งแรกในแต่ละขนาดของข้อมูลขาเข้า จะเรียกเมท็อด `init(int n)`เพื่อเตรียมรายกายจำนวนเต็มแบบสุ่ม

จากนั้นสุ่มหาจำนวนเต็มที่จะถูกเลือกแล้วส่งคืนให้ตัวควบคุมการทดลอง พร้อมรายการจำนวนเต็มแบบสุ่ม

```
import java.util.ArrayList;
import java.util.List;
import java.util.Random;
import jprofile.Profiler;
public class RandomIntList implements InputGenerator{
    private int[] d;
    private List dList;
    private double[] zd;
    private int n;
    public RandomIntList() {
        n = 0;
    }
    public void init(int n) {
        d = new int[n];
        dList = new ArrayList();
        for (int i = 0; i < d.length; i++) {
            d[i] = i;
        }
        Random rgen = new Random();
        //--- Shuffle by exchanging each element randomly
        for (int i = 0; i < d.length; i++) {
            int randomPosition = rgen.nextInt(d.length);
            int temp = d[i];
            d[i] = d[randomPosition];
            d[randomPosition] = temp;
        }
        for (int i = 0; i < d.length; i++){
            dList.add(d[i]);
        }
    }
    @Override
    public Object[] get(int aN) {
        if(n != aN){
            n = aN;
            init(n);
        }
        int i = (int) (n * Profiler.random.nextDouble());
            return new Object[]{dList, new Integer(i)};
    }
}
```

รหัสที่ ข.4 หน่วยผลิตข้อมูลขาเข้า RandomIntList

ข.5 รหัสคำสั่งหน่วยผลิตข้อมูลขาเข้า `RandomZipfIntList` ผลิตข้อมูลในรายกายจำนวนเต็มแบบสุ่ม พร้อมเลือกข้อมูลออกมาเป็นจำนวนเต็ม `x` มีการกระจายแบบ Zipf เมื่อเรียกเมท็อด `get` ครั้งแรกในแต่ละขนาดของข้อมูลขาเข้า จะเรียกเมท็อด `init(int n)`เพื่อเตรียมรายกายจำนวนเต็มแบบสุ่ม และอาร์เรย์ค่าการกระจายแบบ Zipf จากนั้นสุ่มค่าออกมาหนึ่งค่า

เปรียบเทียบกับอาร์เรย์ค่าการกระจายแบบ Zipf หาจำนวนเต็มที่จะถูกเลือกแล้วส่งคืนให้ตัวควบคุมการทดลอง พร้อมรายการ

```
public class RandomZipfIntList implements InputGenerator{
    private int[] d;
    private List dList;
    private double[] zd;
    private int n;
    public RandomZipfIntList() { n = 0; }
    public void init(int n) {
        d = new int[n];
        dList = new ArrayList();
        for (int i = 0; i < d.length; i++) {
            d[i] = i;
        }
        Random rgen = new Random();
        //--- Shuffle by exchanging each element randomly
        for (int i = 0; i < d.length; i++) {
            int randomPosition = rgen.nextInt(d.length);
            int temp = d[i];
            d[i] = d[randomPosition];
            d[randomPosition] = temp;
        }
        for (int i = 0; i < d.length; i++){
            dList.add(d[i]);
        }
        double zn = 0;
        for (int i = 1; i <= n; i++) {
            zn += 1.0 / i;
        }
        zd = new double[n];
        for (int i = 0; i < n; i++) {
            zd[i] = 1.0 / (i + 1) / zn;
        }
        for (int i = 1; i < n; i++) {
            zd[i] += zd[i - 1];
        }
    }
    @Override
    public Object[] get(int aN) {
        if(n != aN){
            n = aN;
            init(n);
        }
        int i;
        double x = Profiler.random.nextDouble();
        for (i = 0; i < zd.length - 1; i++) {
            if (x < zd[i]) {
                break;
            }
        }
        return new Object[]{dList, new Integer(i)};
    }
}
```

รหัสที่ ข.5 หน่วยผลิตข้อมูลขาเข้า RandomZipfIntList

ข.6 รหัสคำสั่งหน่วยแสดงผล `ScatterPlot` แสดงผลการทดลองเป็นกราฟ โดยอาศัยคลังคลาส JFreeChart [21]

```
import java.awt.Color;
import java.awt.event.MouseWheelEvent;
import java.awt.event.MouseWheelListener;
import java.awt.geom.Ellipse2D;
import java.util.ArrayList;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.JComponent;
import jprofile.fitting.FitFunctionInfo;
import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartFrame;
import org.jfree.chart.ChartPanel;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.axis.NumberAxis;
import org.jfree.chart.editor.ChartEditorManager;
import org.jfree.chart.plot.PlotOrientation;
import org.jfree.chart.plot.XYPlot;
import org.jfree.chart.renderer.xy.StandardXYItemRenderer;
import org.jfree.chart.renderer.xy.XYDotRenderer;
import org.jfree.data.xy.DefaultXYDataset;
public class ScatterPlot implements OutputListener {
    private ArrayList x;
    private ArrayList[] y;
    private DefaultXYDataset dataset = new DefaultXYDataset();
    private DefaultXYDataset datasetTime = new DefaultXYDataset();
    private DefaultXYDataset datasetFit = new DefaultXYDataset();
    private String inputGenerator;
    private String name;
    private String methodName;
    private String xtitle;
    private String ytitle;
    private int countNo;
    private String[] countNames;
    private ArrayList[] yMax;
    private ArrayList[] yMin;
    private ArrayList time;
    private ArrayList fitFunctionList;
    private boolean commonOutput;
    private double maxY;
    static Logger logger = Logger.getLogger(ScatterPlot.class.getName());
    @Override
    public void name(String aName) {
        name = aName;
    }
    @Override
    public void xtitle(String aXtitle) {
        xtitle = aXtitle;
    }
    @Override
    public void ytitle(String aYtitle) {
        ytitle = aYtitle;
    }
    @Override
```

```
    public void fitFunction(ArrayList<FitFunctionInfo> afitFunctionList) {
        fitFunctionList = afitFunctionList;
    }
    @Override
    public void newInput(String aInputGeneratorName) {
        //prepair data for new input
        if (x != null) {
            genDataset();
        }
        logger.log(Level.INFO, "init ScatterPlot");
        inputGenerator = aInputGeneratorName;
        x = new ArrayList();
        logger.log(Level.INFO, "init ScatterPlot countNo = " + countNo);
        y = new ArrayList[countNo];
        yMax = new ArrayList[countNo];
        yMin = new ArrayList[countNo];
        for (int indCnt = 0; indCnt < countNo; indCnt++) {
            y[indCnt] = new ArrayList();
            yMax[indCnt] = new ArrayList();
            yMin[indCnt] = new ArrayList();
        }
        time = new ArrayList();
    }
    @Override
    public void newData(double n, double[] avg, double[] max, double[] min){
        x.add(n);
        if (avg != null) {
            for (int indCnt = 0; indCnt < countNo; indCnt++) {
                y[indCnt].add(avg[indCnt]);
            }
        }
        if (max != null) {
            for (int indCnt = 0; indCnt < countNo; indCnt++) {
                yMax[indCnt].add(max[indCnt]);
            }
        }

        if (min != null) {
            for (int indCnt = 0; indCnt < countNo; indCnt++) {
                yMin[indCnt].add(min[indCnt]);
            }
        }
    }
    @Override
    public void finish() {
        if (x != null) {
            genDataset();
        }
        JFreeChart chart = ChartFactory.createScatterPlot(
                name,
                xtitle,
                ytitle,
                dataset,
                PlotOrientation.VERTICAL,
                true, // legend?
                true, // tooltips?
```

```
                false // URLs?
                );
        ChartEditorManager.getChartEditor(chart);
        final XYPlot plotBg = chart.getXYPlot();
        plotBg.setBackgroundPaint(Color.white);
        plotBg.setDomainGridlinePaint(Color.BLACK);
        plotBg.setRangeGridlinePaint(Color.BLACK);
        //fit line
        int dataSetIndex = 0;
        if (datasetFit.getSeriesCount() > 0) {
            dataSetIndex++;
            final XYPlot plot = chart.getXYPlot();
            final StandardXYItemRenderer renderer2 =
                                            new StandardXYItemRenderer();
            renderer2.setDefaultEntityRadius(2);
            plot.setDataset(dataSetIndex, datasetFit);
            plot.setRenderer(dataSetIndex, renderer2);
        }
        //chart time
        if (datasetTime.getSeriesCount() > 0) {
            dataSetIndex++;
            final XYPlot plot = chart.getXYPlot();
            final NumberAxis axis2 = new NumberAxis("Time(ns.)");
            final XYDotRenderer renderer2 = new XYDotRenderer();
            renderer2.setDotHeight(5);
            renderer2.setDotWidth(5);
            renderer2.setAutoPopulateSeriesOutlinePaint(true);
            renderer2.setBaseShape(new Ellipse2D.Double(-4, -4, 7, 7));
            plot.setRenderer(dataSetIndex, renderer2);
            axis2.setAutoRangeIncludesZero(false);
            plot.setRangeAxis(1, axis2);
            plot.setDataset(dataSetIndex, datasetTime);
            plot.mapDatasetToRangeAxis(dataSetIndex, 1);
        }
        ChartFrame frame = new ChartFrame(name, chart);
        ChartPanel panel = frame.getChartPanel();
        addListener(panel);
        frame.pack();
        frame.setVisible(true);
    }
    @Override
    public void timer(double value) {
        time.add(value);
    }
    private void genDataset() {
        double[][] data = null;
        double[][] dataMax = null;
        double[][] dataMin = null;
        String serieName = "";
        if (commonOutput) {
            serieName = methodName + "_";
        }
        serieName = serieName + inputGenerator;
        for (int indCnt = 0; indCnt < countNo; indCnt++) {
            String countName = countNames[indCnt];
            if (!countName.equalsIgnoreCase("")) {
```

```
                countName = "_" + countName;
            }
            if (fitFunctionList != null && fitFunctionList.size() > 0) {
                FitFunctionInfo fitFunctionInfo =
                            (FitFunctionInfo) fitFunctionList.get(indCnt);
                //fitdata
                double[] fitDataY = fitFunctionInfo.getY();
                int fitDisplay = 0;
                for (int ind = 0; ind < x.size(); ind++) {
                     if (fitDataY[ind] < maxY) {
                         fitDisplay++;
                     }else{
                         break;
                     }
                }
                double[][] fitData = new double[2][fitDisplay];
                for (int ind = 0; ind < fitDisplay; ind++) {
                    if (fitDataY[ind] < maxY) {
                        fitData[0][ind] = (Double) x.get(ind);
                        fitData[1][ind] = fitDataY[ind];
                    }
                }
                datasetFit.addSeries(serieName + countName + "_FIT",
                                     fitData);
            }
            //Average
            if (y[indCnt].size() > 0) {
                int avgDisplay = 0;
                for (int ind = 0; ind < x.size(); ind++) {
                     if ((Double) y[indCnt].get(ind) < maxY) {
                         avgDisplay++;
                     }else{
                         break;
                     }
                }
                data = new double[2][avgDisplay];
                for (int ind = 0; ind < avgDisplay; ind++) {
                     data[0][ind] = (Double) x.get(ind);
                     data[1][ind] = (Double) y[indCnt].get(ind);
                }
                dataset.addSeries(serieName + countName, data);
            }
            //Max
            if (yMax[indCnt].size() > 0) {

                int maxDisplay = 0;
                for (int ind = 0; ind < x.size(); ind++) {
                     if ((Double) yMax[indCnt].get(ind) < maxY) {
                         maxDisplay++;
                     }else{
                         break;
                     }
                }
                dataMax = new double[2][maxDisplay]
                for (int ind = 0; ind < maxDisplay; ind++) {
                    if ((Double) yMax[indCnt].get(ind) < maxY) {
```

```
                    dataMax[0][ind] = (Double) x.get(ind);
                    dataMax[1][ind] = (Double) yMax[indCnt].get(ind);
                }
            }
            dataset.addSeries(serieName + countName + "_MAX", dataMax);
        }
        //Min
        if (yMin[indCnt].size() > 0) {
            int minDisplay = 0;
            for (int ind = 0; ind < x.size(); ind++) {
                 if ((Double) yMin[indCnt].get(ind) < maxY) {
                     minDisplay++;
                 }else{
                     break;
                 }
            }
            dataMin = new double[2][minDisplay];
            for (int ind = 0; ind < minDisplay; ind++) {
                if ((Double) yMin[indCnt].get(ind) < maxY) {
                    dataMin[0][ind] = (Double) x.get(ind);
                    dataMin[1][ind] = (Double) yMin[indCnt].get(ind);
                }
            }
            dataset.addSeries(serieName + countName + "_MIN", dataMin);
        }
        int dataMinSize = Math.min(x.size(), yMin[indCnt].size());
        if (dataMinSize > 0) {
            dataMin = new double[2][x.size()];
            for (int ind = 0; ind < x.size(); ind++) {
                if ((Double) yMin[indCnt].get(ind) < maxY) {
                    dataMin[0][ind] = (Double) x.get(ind);
                    dataMin[1][ind] = (Double) yMin[indCnt].get(ind);
                }
            }
            dataset.addSeries(serieName + countName + "_MIN", dataMin);
        }
    }
    if (time.size() > 0) {
        data = new double[2][x.size()];
        for (int ind = 0; ind < x.size(); ind++) {
            data[0][ind] = (Double) x.get(ind);
            data[1][ind] = (Double) time.get(ind);
        }
        datasetTime.addSeries(serieName + "_TIME", data);
    }
    x = null; y = null;
    yMax = null; yMin = null;
    time = null;
}
protected void addListener(ChartPanel chartPanel) {
    chartPanel.addMouseWheelListener(new MouseWheelListener() {

        @Override
        public void mouseWheelMoved(MouseWheelEvent e) {
            if (e.getScrollType() != MouseWheelEvent.WHEEL_UNIT_SCROLL){
                return;
```

```
                }
                if (e.getWheelRotation() < 0) {
                    increaseZoom((ChartPanel) e.getComponent(), true);
                } else {
                    decreaseZoom((ChartPanel) e.getComponent(), true);
                }
            }
            public synchronized void increaseZoom(JComponent chart,
                                                  boolean saveAction) {
                ChartPanel ch = (ChartPanel) chart;
                zoomChartAxis(ch, true);
            }
            public synchronized void decreaseZoom(JComponent chart,
                                                 boolean saveAction) {
                ChartPanel ch = (ChartPanel) chart;
                zoomChartAxis(ch, false);
            }
            private void zoomChartAxis(ChartPanel chartP, boolean increase){
                int width = chartP.getMaximumDrawWidth() –
                            chartP.getMinimumDrawWidth();
                int height = chartP.getMaximumDrawHeight() -
                            chartP.getMinimumDrawWidth();
                if (increase) {
                    chartP.zoomInBoth(width / 2, height / 2);
                } else {
                    chartP.zoomOutBoth(width / 2, height / 2);
                }
            }
        });
    }
    @Override
    public void commonOutput(boolean aCommonOutput) {
        commonOutput = aCommonOutput;
    }
    @Override
    public void newMethod(String aMethodName, int aCountNo,
                          String[] aCountNames) {
        if (x != null) {
            genDataset();
        }
        methodName = aMethodName;
        countNo = aCountNo;
        countNames = aCountNames;
    }
    @Override
    public void maxY(Double value) {
        maxY = value;
    }
}
```

รหัสที่ ข.6 หน่วยแสดงผล ScatterPlot

ข.7 รหัสคำสั่งหน่วยแสดงผล `ExcelFile` แสดงผลการทดลองเป็นแฟ้มตาราง ทำงานในรูปแบบ XLS ของไมโครซอฟต์เอ็กซ์เซล (Excel) โดยอาศัยคลังคลาส POI-HSSF [22]

```
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.ArrayList;
import java.util.logging.Level;
import java.util.logging.Logger;
import jprofile.fitting.FitFunctionInfo;
import jprofile.util.FileUtil;
import org.apache.poi.hssf.usermodel.HSSFRichTextString;
import org.apache.poi.hssf.usermodel.HSSFRow;
import org.apache.poi.hssf.usermodel.HSSFSheet;
import org.apache.poi.hssf.usermodel.HSSFWorkbook;
public class ExcelFile implements OutputListener{
    private  ArrayList x;
    private  ArrayList[] y;
    private  ArrayList[] yMax;
    private  ArrayList[] yMin;
    private  ArrayList time;
    private  ArrayList fitFunctionList;
    private  String name;
    private  String xtitle;
    private  String ytitle;
    private  int countNo;
    private  String[] countNames;
    private  String inputGenerator;
    private  String methodName;
    private boolean commonOutput;
    private double maxY;
    private  ArrayList<DatasetEntry> dataset =
                                            new ArrayList<DatasetEntry>();
    static Logger logger = Logger.getLogger(ScatterPlot.class.getName());
    @Override
    public void name(String aName) {
        name = aName;
    }
    @Override
    public void xtitle(String aXtitle) {
        xtitle = aXtitle;
    }
    @Override
    public void ytitle(String aYtitle) {
        ytitle = aYtitle;
    }
    @Override
    public void newInput(String aInputGeneratorName) {
        //prepair data for new input
        if (x != null) {
            genDataset();
        }
        logger.log(Level.INFO, "init Excel File");
        inputGenerator = aInputGeneratorName;
        x = new ArrayList();
        logger.log(Level.INFO, "init Excel countNo = " + countNo);
        y = new ArrayList[countNo];
```

```
        yMax = new ArrayList[countNo];
        yMin = new ArrayList[countNo];
        for (int indCnt = 0; indCnt < y.length; indCnt++) {
            y[indCnt] = new ArrayList();
            yMax[indCnt] = new ArrayList();
            yMin[indCnt] = new ArrayList();
        }
        time = new ArrayList();
    }
    @Override
    public void newData(double n,double[] avg,double[] max,double[] min) {
        x.add(n);
        if (avg != null) {
            for (int indCnt = 0; indCnt < countNo; indCnt++) {
                y[indCnt].add(avg[indCnt]);
            }
        }
        if (max != null) {
            for (int indCnt = 0; indCnt < countNo; indCnt++) {
                yMax[indCnt].add(max[indCnt]);
            }
        }
        if (min != null) {
            for (int indCnt = 0; indCnt < countNo; indCnt++) {
                yMin[indCnt].add(min[indCnt]);
            }
        }
    }
    @Override
    public void finish() {
        if (x != null) {
            genDataset();
        }
        //create file
        HSSFWorkbook wb = new HSSFWorkbook();
        HSSFSheet sheet = wb.createSheet(name);
        HSSFRow rowName = sheet.createRow(0);
        for (int datasetInd = 0; datasetInd < dataset.size(); datasetInd++){
            DatasetEntry entry = dataset.get(datasetInd);
            // Create a row and put some cells in it. Rows are 0 based
            // Create a cell and put a value in it.
            HSSFRichTextString text = new HSSFRichTextString(entry.name);
            rowName.createCell(datasetInd).setCellValue(text);
            ArrayList dataList = entry.getData();
            int ind = 1;
            for(Object obj : dataList ){
                HSSFRow row = sheet.getRow(ind);
                if(row == null){
                    row = sheet.createRow(ind);
                }
                row.createCell(datasetInd).setCellValue((Double) obj);
                ind++;
            }
        }
        for (int ind = 0; ind < dataset.size(); ind++) {
            sheet.autoSizeColumn((short) ind);
```

```
        }
        //delete file
        FileUtil fileUtil = new FileUtil();
        String tempName =  new String(name);
        int fileIndex = 1;
        FileOutputStream fileOut;
        while (true) {
            try {
                fileUtil.deleteFile(name + ".xls");
                fileOut = new FileOutputStream(name + ".xls");
                wb.write(fileOut);
                fileOut.close();
                break;
            } catch (IOException ex) {
               Logger.getLogger(ExcelFile.class.getName()).log(Level.SEVERE,
                         "Create Excel File "+name + ".xls Fail.", ex);
            }
            name = tempName + fileIndex;
            fileIndex++;
        }
    }
    @Override
    public void fitFunction(ArrayList<FitFunctionInfo> afitFunctionList) {
        fitFunctionList = afitFunctionList;
    }
    @Override
    public void timer(Double value) {
           time.add(value);
    }
    private void genDataset(){
        String serieName = "";
        if (commonOutput) {
            serieName = methodName + "_";
        }
         serieName = serieName + inputGenerator;
        if(dataset.size() == 0){
            dataset.add(new DatasetEntry(xtitle, x));
        }
        for (int indCnt = 0; indCnt < y.length; indCnt++) {
            String countName = countNames[indCnt];
            if (!countName.equalsIgnoreCase("")) {
                countName = "_" + countName;
            }
            String fitFunctionName = "";
            if (fitFunctionList != null && fitFunctionList.size() > 0) {

                FitFunctionInfo fitFunctionInfo =
                            (FitFunctionInfo) fitFunctionList.get(indCnt);
                fitFunctionName = "_" + fitFunctionInfo.getFunctionName();
                //fitdata
                dataset.add(new DatasetEntry(serieName + countName +
                fitFunctionName + "_FIT", fitFunctionInfo.getArrayListY()));
            }
            if (y[indCnt].size() > 0) {
                for (int indMax = 0; indMax < y.length; indMax++) {
                    dataset.add(new DatasetEntry(serieName + countName,
                                                 y[indCnt]));
```

```
                }
            }
            if (yMax[indCnt].size() > 0) {
                for (int indMax = 0; indMax < yMax.length; indMax++) {
                    dataset.add(new DatasetEntry(serieName + countName +
                                "_MAX", yMax[indMax]));
                }
            }
            if (yMin[indCnt].size() > 0) {
                for (int indMax = 0; indMax < yMin.length; indMax++) {
                    dataset.add(new DatasetEntry(serieName + countName +
                                "_MIN", yMin[indMax]));
                }
            }
        }
        if(time.size() > 0){
            dataset.add(new DatasetEntry(serieName+"_TIME", time));
        }
        x = null;
        y = null;
        yMax = null;
        yMin = null;
        time = null;
    }
    @Override
    public void commonOutput(boolean aCommonOutput) {
       commonOutput = aCommonOutput;
    }
    @Override
    public void newMethod(String aMethodName, int aCountNo,
                          String[] aCountNames) {
        if (x != null) {
            genDataset();
        }
        methodName = aMethodName;
        countNo = aCountNo;
        countNames = aCountNames;
    }
    @Override
    public void maxY(Double value) {
         maxY = value;
    }
    private class DatasetEntry{
        String name;
        ArrayList data;
        public DatasetEntry(String name, ArrayList data) {
            this.name = name;
            this.data = data;
        }
        public ArrayList getData() {
            return data;
        }
        public void setData(ArrayList data) {
            this.data = data;
        }
        public String getName() {
```

```
            return name;
        }
        public void setName(String name) {
            this.name = name;
        }
    }
}
```

รหัสที่ ข.7 หน่วยแสดงผล ExcelFile

# ประวัติผู้เขียนวิทยานิพนธ์

นายวิริยะ นิลละออ เกิดวันที่ 20 มิถุนายน พ.ศ. 2524 ที่จังหวัดอุดรธานี สำเร็จการศึกษาหลักสูตรวิทยาศาสตรบัณฑิต (วท.บ.) สาขาวิชาวิทยาการคอมพิวเตอร์ ภาควิชาวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์ มหาวิทยาลัยขอนแก่น ในปีการศึกษา 2546 และเข้าศึกษาต่อในหลักสูตรวิทยาศาสตรมหาบัณฑิต สาขาวิชาวิทยาศาสตร์คอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์ จุฬาลงกรณ์มหาวิทยาลัย เมื่อปีการศึกษา 2550

งานวิจัยชิ้นนี้ได้รับการเผยแพร่และตีพิมพ์ในงาน The 2nd Nationnal Conference on Information Technology 2008 (NCIT2008) จัดโดย มหาวิทยาลัยรังสิต ในระหว่างวันที่ 6 - 7 พฤษจิกายน 2551 ณ โรงแรมแกรนด์ เมอร์เคียว ฟอร์จูน กรุงเทพฯ