ระบบการวิเคราะห์อัลกอริทึมเชิงทดลองด้วยการวัดคำสั่งการทำงาน

นายธนินทร์ กระจังทอง

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต
สาขาวิชาวิทยาศาสตร์คอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2554

EXPERIMENTAL ANALYSIS OF ALGORITHM SYSTEM USING INSTRUCTION INSTRUMENTATION

Mr. Tanin Krajangthong

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 2011

หัวข้อวิทยานิพนธ์	ระบบการวิเคราะห์อัลกอริทึมเชิงทดลองด้วยการวัดคำสั่งการทำงาน
โดย	นายธนินทร์ กระจังทอง
สาขาวิชา	วิทยาศาสตร์คอมพิวเตอร์
อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก	รองศาสตราจารย์ ดร.สมชาย ประสิทธิ์จูตระกูล

คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย อนุมัติให้นับวิทยานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญามหาบัณฑิต

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

คณะกรรมการสอบวิทยานิพนธ์

…… ประธานกรรมการ
(อาจารย์ ดร.นัทที นิภานันท์)

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

…… กรรมการภายนอกมหาวิทยาลัย
(อาจารย์ ดร.ณรงค์เดช กีรติพรานนท์)

ธนินทร์ กระจังทอง : ระบบการวิเคราะห์อัลกอริทึมเชิงทดลองด้วยการวัดคำสั่งการทำงาน. (Experimental Analysis of Algorithm System using Instruction Instrumentation) อ.ที่ปรึกษาวิทยานิพนธ์หลัก : รศ.ดร.สมชาย ประสิทธิ์จูตระกูล, 64 หน้า.

การวิเคราะห์อัลกอริทึมมีวัตถุประสงค์เพื่อนับจำนวนครั้งการทำงานในแต่ละบรรทัดของรหัสต้นฉบับระหว่างทำการทดลอง เพื่อศึกษาประสิทธิภาพเชิงเวลาการทำงานของอัลกอริทึม วิทยานิพนธ์นี้นำเสนอรูปแบบบริการเพื่อการวิเคราะห์อัลกอริทึมเชิงทดลองนำเสนอผ่านเว็บและเว็บเซอร์วิส รองรับรหัสอัลกอริทึมภาษาจาวา สามารถกำหนดรูปแบบการทดลองได้แก่ ขนาดของข้อมูลขาเข้า และลักษณะของข้อมูลขาเข้า และเก็บข้อมูลการทดลองโดยใช้รูปแบบต้นไม้ไวยากรณ์ที่เป็นนามธรรม(AST) ในการสำรวจต้นไม้ และแทรกคำสั่งในการนับการทำงานของแต่ละบรรทัดของรหัสต้นฉบับอย่างอัตโนมัติ จากนั้นจึงแปลงรูปแบบต้นไม้กลับเป็นรหัสต้นฉบับที่พร้อมสำหรับการทดลอง โดยผลลัพธ์จะนำเสนอในรูปแบบกราฟเส้นเพื่อแสดงอัตราการเติบโตของอัลกอริทึม และฮิสโตแกรมเพื่อสะท้อนปริมาณการทำงานในแต่ละบรรทัดของรหัสต้นฉบับ วิทยานิพนธ์นี้สามารถนำไปใช้ได้กับการเรียนการสอนในเนื้อหารายวิชาที่เกี่ยวกับการวิเคราะห์อัลกอริทึม

ภาควิชา วิศวกรรมคอมพิวเตอร์ ลายมือชื่อนิสิต .....

สาขาวิชา วิทยาศาสตร์คอมพิวเตอร์ ลายมือชื่ออ.ที่ปรึกษาวิทยานิพนธ์ .....

ปีการศึกษา 2554

5271426021 : MAJOR COMPUTER SCIENCE

KEYWORDS : ANALYSIS OF ALGORITHM / EXPERIMENTAL ANALYSIS

TANIN KRAJANGTHONG : EXPERIMENTAL ANALYSIS OF ALGORITHM SYSTEM USING INSTRUCTION INSTRUMENTATION. ADVISOR : ASSOC. PROF. DR. SOMCHAI PRASITJUTRAKUL Ph.D., 64 pp.

Analysis of algorithm objective is to count the number of times each source code instruction gets executed during the experiments to study the efficiency of the algorithm. This research presents a service for experimental analysis of algorithms presented by web application and web service support algorithms written as source codes in Java programming language. Experimental parameters can be configured such as range of input sizes and input characteristics. This is done by source-code instrumentation that parses the source code to obtain its associated abstract syntax tree (AST), traversing the tree, inserting extra counting instructions at instruction nodes and finally transforming the tree back into an instrumented source code ready for experiments. Experimental results are shown as a scatter plot of running time versus input size along with their correlation using curve fitting. In addition, an instruction-execution-count histogram is also shown adjacent to the source code for better visualization. This system is used effectively in teaching algorithm analysis courses.

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

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

Academic Year : 2011

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้สำเร็จลุล่วงไปด้วยดีด้วยความช่วยเหลือจากรองศาสตราจารย์ ดร.สมชาย ประสิทธิ์จูตระกูล ซึ่งเป็นอาจารย์ที่ปรึกษาวิทยานิพนธ์ ที่ซึ่งได้มอบความรู้ คำแนะนำ ตรวจทานเพื่อแก้ไขส่วนบกพร่องของงานวิจัย ตลอดจนการตรวจทานแก้ไขวิทยานิพนธ์ให้มีความสมบูรณ์ นอกจากนี้ผู้เขียนยังได้รับความกรุณาจากอาจารย์ ดร.นัทที นิภานันท์ ผู้ซึ่งเป็นประธานกรรมการสอบวิทยานิพนธ์ รวมถึงอาจารย์ ดร.ณรงค์เดช กีรติพรานนท์ อาจารย์กรรมการสอบวิทยานิพนธ์ผู้ทรงคุณวุฒิจากภายนอก ที่ได้ให้คำแนะนำ รวมทั้งข้อเสนอแนะต่างๆที่เป็นประโยชน์เพื่อนำมาใช้ปรับปรุงวิทยานิพนธ์ให้เกิดความสมบูรณ์มากยิ่งขึ้น

ผู้เขียนขอกราบขอบพระคุณบิดา มารดา ที่ได้สนับสนุนด้านทุนทรัพย์ในการศึกษาและคอยเป็นกำลังใจให้เสมอมา รวมทั้งเพื่อนสนิทของข้าพเจ้าที่คอยให้กำลังใจและเป็นแรงบันดาลใจให้ข้าพเจ้าเสมอมา

สารบัญ

หน้า

สารบัญตาราง

หน้า

สารบัญภาพ

หน้า

หน้า

สารบัญรหัส

หน้า

บทที่ 1

บทนำ

1.1. ความเป็นมาและความสำคัญของปัญหา

การวิเคราะห์อัลกอริทึมมีวัตถุประสงค์เพื่อศึกษาประสิทธิภาพเชิงเวลาการทำงานของอัลกอริทึม ว่ามีความสัมพันธ์อย่างไรกับข้อมูลขาเข้า [1] ที่มีขนาด, ปริมาณ และลักษณะของข้อมูลขาเข้าที่แตกต่างกัน โดยการวิเคราะห์อาจเป็นการวิเคราะห์กรณีที่ช้าที่สุด หรือค่าเฉลี่ยที่เกิดขึ้น โดยอาศัยการเขียนความสัมพันธ์หรือฟังก์ชันทางคณิตศาสตร์เพื่อแทนจำนวนครั้งที่มากที่สุด หรือเฉลี่ย หรืออาจจะนับคำสั่งตัวแทน และอาศัยการวิเคราะห์เชิงเส้นกำกับ (Asymptotic Analysis) เพื่อให้ได้ผลสะท้อนอัตราการเติบโตของฟังก์ชันเพื่อใช้เปรียบเทียบประสิทธิภาพเชิงเวลาของอัลกอริทึมต่างๆ

การวิเคราะห์อัลกอริทึมต่างๆที่ได้กล่าวข้างต้นนั้น ผู้วิเคราะห์อาจใช้วิธีการวิเคราะห์เชิงทดลอง [2] โดยวิธีการเขียนโปรแกรมตามอัลกอริทึมที่สนใจ เพื่อทดลองการทำงานและเก็บข้อมูลการทำงานตลอดการทำงานของโปรแกรม โดยงานวิจัยนี้นำเสนอวิธีการแทรกคำสั่งในรหัสต้นฉบับเพื่อทำการเก็บข้อมูลและอาศัยการเขียนคำอธิบายประกอบ(Annotation) [3] กำกับ เมท็อดด้วยภาษาจาวาเพื่อบรรยายรูปแบบการทดลอง และใช้วิธีการนับคำสั่งทุกคำสั่งเพื่อใช้ทดสอบการทำงานของแต่ละอัลกอริทึม ทำการทดลองกับข้อมูลขาเข้าที่หลากหลาย เพื่อใช้ในการเปรียบเทียบหาความสัมพันธ์ของเวลาการทำงานกับขนาด, ปริมาณ และลักษณะของข้อมูลขาเข้ารวมทั้งรูปแบบการแสดงผลและการบันทึกผลการทดลอง ตัวควบคุมจะป้อนข้อมูลขาเข้า บันทึกผล และทำการวนทำการทดลองตามรูปแบบที่กำหนดไว้ และแสดงผลการทดลองโดยการกรองคำสั่งตัวแทนที่ผ่านการนับทุกคำสั่ง เพื่อใช้เป็นตัวแทนในการวิเคราะห์เปรียบเทียบประสิทธิภาพเชิงเวลาของแต่ละอัลกอริทึมอย่างอัตโนมัติ และการแสดงผลจะอยู่ในรูปแบบกราฟเส้น และกราฟฮิสโตแกรมเพื่อให้เห็นภาพความถี่การทำงานในแต่ละบรรทัดของรหัสต้นฉบับได้ชัดเจนมากยิ่งขึ้น

ภาพที่ 1.1 ภาพรวมการทำงานของระบบระบบการวิเคราะห์อัลกอริทึมเชิงทดลองด้วยการวัดคำสั่งการทำงาน

ภาพที่ 1.1 แสดงให้เห็นการทำงานของระบบการวิเคราะห์อัลกอริทึมเชิงทดลองด้วยการวัดคำสั่งการทำงาน การทำงานจะเริ่มจากผู้ใช้งานนำรหัสต้นฉบับของอัลกอริทึมที่กำกับการบรรยายรูปแบบการทดลอง (หมายเลข 1) ส่งเข้าไปในระบบและทำการทดลองอย่างอัตโนมัติ (หมายเลข 2) โดยที่ผู้ใช้ไม่ต้องระบุบรรทัดที่เป็นตัวแทนของแต่ละอัลกอริทึม รวมทั้งไม่จำเป็นต้องทราบส่วนที่ทำงานหนักที่สุดของแต่ละอัลกอริทึม เมื่อผลลัพธ์จากการทดลองที่ได้ผ่านระบบ (หมายเลข 3) ผู้ใช้จะทราบบรรทัดที่เป็นตัวแทนของอัลกอริทึม โดยจะสะท้อนอัตราการเติบโตในรูปแบบของกราฟ และทราบปริมาณความถี่ของการทำงานของอัลกอริทึมในแต่ละบรรทัดผ่านการแสดงผลลัพธ์ในรูปแบบฮิสโตแกรม

1.2. วัตถุประสงค์ของการวิจัย

งานวิจัยนี้มีวัตถุประสงค์เพื่อออกแบบและพัฒนาเครื่องมือในการวิเคราะห์อัลกอริทึมเชิงทดลอง โดยใช้วิธีการแทรกคำสั่งนับในรหัสต้นฉบับทุกคำสั่ง ทำการทดลองผ่านเครื่องมือควบคุมการทดลอง JProfile101 [4] และคัดกรองแสดงผลเฉพาะคำสั่งตัวแทนเพื่อใช้ในการเปรียบเทียบเชิงเวลาของแต่ละอัลกอริทึมอย่างอัตโนมัติ เป็นการสะท้อนให้เห็นถึงการทำงานของอัลกอริทึมโดยผู้วิเคราะห์ไม่จำเป็นต้องระบุบรรทัดที่เป็นตัวแทนของอัลกอริทึมนั้นๆ ทำให้ไม่เกิดความผิดพลาดอันเนื่องมาจากผู้ใช้ระบุบรรทัดไม่ถูกต้อง ทำให้การศึกษาการทำงานของอัลกอริทึม ผ่านผลลัพธ์การทดลองเป็นไปอย่างมีประสิทธิภาพ

1.3. ขอบเขตของการวิจัย

1.3.1. พัฒนาระบบด้วยภาษาจาวา

1.3.2. ระบบสามารถแทรกคำสั่งนับในรหัสต้นฉบับได้อย่างอัตโนมัติ

1.3.3. ระบบสามารถแสดงผลการทดลองในรูปแบบของกราฟเส้นและฮิสโตแกรม

1.3.4. ระบบอาศัยต้นไม้ไวยากรณ์นามธรรม (Abstract Syntax Tree) ช่วยในการแทรกคำสั่งนับกำกับคำสั่งในรหัสต้นฉบับ

1.3.5. ระบบอาศัย JProfile101 ในการผลิตข้อมูลขาเข้า โดยใช้วิธีการระบุ Annotation ของจาวาในการระบุลักษณะการทดลอง และทำการทดลองอย่างอัตโนมัติ

1.4. คำจำกัดความที่ใช้ในการวิจัย

1.4.1. ต้นไม้ไวยากรณ์นามธรรม (Abstract Syntax Tree) คือ ตัวแทนโครงสร้างนามธรรมของรหัสต้นฉบับ

1.4.2. รหัสต้นฉบับ (Source Code) คือ ข้อความที่เป็นชุดที่ถูกเขียนขึ้น และสามารถอ่านและเข้าใจได้ ใช้สำหรับภาษาโปรแกรม เพื่อใช้ในการประมวลผลสำหรับคอมพิวเตอร์โดยคอมไพเลอร์สำหรับโปรแกรมนั้น

1.4.3. ฮิสโตแกรม (Histogram) คือ กราฟแท่งแบบเฉพาะใช้แสดงจำนวนครั้งการทำงานกำกับในแต่ละบรรทัดของรหัสต้นฉบับ

1.4.4. คำอธิบายประกอบ (Annotation) คือ สิ่งที่ใช้สำหรับอธิบายประกอบส่วนต่างๆ ในรหัสต้นฉบับ รวมทั้งใช้กำกับการทำงานของส่วนต่างๆ ของรหัสต้นฉบับ

1.4.5. รหัสดำเนินการ (Operation Code) หมายถึงส่วนของคำสั่งในภาษาเครื่อง (Machine Language) หรือภาษาแอสเซมบลี (Assembly Language) ที่เป็นคำสั่งให้เครื่องปฏิบัติ

1.5. ประโยชน์ที่คาดว่าจะได้รับ

งานวิจัยนี้นำเสนอระบบที่สามารถแทรกคำสั่งเพื่อใช้ในการนับความถี่การทำงานของคำสั่งต้นฉบับทุกคำสั่ง โดยผู้ใช้งานระบบไม่ต้องระบุบรรทัดคำสั่งในการวิเคราะห์อัลกอริทึม ทำให้ลดโอกาสพบความผิดพลาดในการระบุคำสั่งตัวแทนในกรณีที่ผู้ใช้งานไม่มีความชำนาญ

เพียงพอ ระบบสามารถบอกบรรทัดที่ใช้เป็นตัวแทนในการวิเคราะห์อัลกอริทึม เพื่อนำผลลัพธ์ที่ได้ไปวิเคราะห์การทำงานหรืออัตราการเติบโตของแต่ละอัลกอริทึมต่อไป

ระบบที่พัฒนาขึ้นสามารถช่วยให้มองเห็นจุดอ่อนและข้อบกพร่องของรหัสต้นฉบับ เพราะระบบสามารถแสดงความถี่การทำงานของแต่ละคำสั่งในรหัสต้นฉบับ เพื่อให้ผู้ใช้นำผลลัพธ์ไปปรับปรุงและพัฒนารหัสต้นฉบับที่ใช้ทดลองนั้นๆ ต่อไป

1.6. **วิธีดำเนินการวิจัย**

1.6.1. ศึกษาและทดสอบแนวความคิดและทฤษฎีของงานวิจัยที่เกี่ยวข้อง

1.6.2. ทดลองแนวทางและวิธีการของงานวิจัยต่างๆ เพื่อแทรกรหัสคำสั่งนับลงในรหัสต้นฉบับทุกบรรทัด

1.6.3. เปรียบเทียบแนวทางและวิธีการของงานวิจัยต่างๆ ภายหลังการทดลองเพื่อนำวิธีการที่เหมาะสมที่สุดมาใช้ในงานวิจัย

1.6.4. ทดสอบ วิเคราะห์ ปรับปรุงแก้ไขงานวิจัย ให้อยู่ในแนวทางที่ต้องการ

1.6.5. นำเสนอผลลัพธ์ของงานวิจัยที่ได้ในรูปแบบต่างๆ เช่น การแสดงผลในรูปแบบกราฟ หรือฮิสโตแกรม

1.6.6. สรุปผลงานวิจัย

1.7. **ลำดับขั้นตอนในการเสนอผลการวิจัย**

วิทยานิพนธ์นี้แบ่งเนื้อหาทั้งหมด 5 บท โดยแต่ละบทประกอบไปด้วยเนื้อหาดังต่อไปนี้

บทที่ 1 นำเสนอ ความเป็นมาและความสำคัญของปัญหา, วัตถุประสงค์ของการวิจัย, ขอบเขตของการวิจัย, คำจำกัดความที่ใช้ในการวิจัย, ประโยชน์ที่คาดว่าจะได้รับ และวิธีดำเนินการวิจัย

บทที่ 2 อธิบาย

1. ทฤษฎีที่เกี่ยวข้อง เช่น การวิเคราะห์อัลกอริทึม, โพรไฟล์เลอร์ และต้นไม้ไวยากรณ์ที่เป็นนามธรรม

2. เอกสารและงานวิจัยที่เกี่ยวข้อง เช่น Algorithm Analysis, JP และ ByCounter

บทที่ 3 ประกอบด้วย

1. ภาพรวมและขั้นตอนการทำงานของระบบ เช่น การแทรกคำสั่งนับทุกคำสั่ง, การทดลองผ่าน JProfile101 และการประมวลผลข้อมูลที่ได้จากการทดลองพร้อมทั้งแสดงผล

2. การพัฒนาการเข้าใช้งานระบบ เช่น การใช้งานระบบผ่านหน้าเว็บ, การใช้งานระบบผ่านบริการเว็บเซอร์วิส

บทที่ 4 นำเสนอ ตัวอย่างและผลลัพธ์การทดลอง เช่น ตัวอย่างอัลกอริทึมทางคณิตศาสตร์ในการจัดเรียงข้อมูลแบบ Quicksort, ตัวอย่างอัลกอริทึมทางคณิตศาสตร์ในการจัดเรียงข้อมูลแบบ Merge Sort, ตัวอย่างอัลกอริทึมทางคณิตศาสตร์ในการจัดเรียงข้อมูลแบบ Insertion Sort, ตัวอย่างอัลกอริทึมทางคณิตศาสตร์ในการจัดเรียงข้อมูลแบบ Bubble Sort และ, ตัวอย่างอัลกอริทึมทางคณิตศาสตร์ในการหาค่าสูงที่สุด Find Max

บทที่ 5 สรุปผลการวิจัย

บทที่ 2
เอกสารและงานวิจัยที่เกี่ยวข้อง

2.1. ทฤษฎีที่เกี่ยวข้อง

2.1.1. การวิเคราะห์อัลกอริทึม

การวิเคราะห์อัลกอริทึม [1] มีเป้าหมายเพื่อวัดประสิทธิภาพของแต่ละอัลกอริทึม คือการประมาณค่าทรัพยากรที่จำเป็นในการทำงาน เช่น เวลา หรือหน่วยความจำ โดยประสิทธิภาพ หรือความซับซ้อนของอัลกอริทึมจะวัดจากความสัมพันธ์ของข้อมูลขาเข้ากับเวลาที่ใช้ไปในการทำงาน หรือหน่วยความจำที่ใช้ไป ในงานวิจัยนี้จะเน้นเฉพาะเรื่องของเวลา โดยอาศัยการนับจำนวนครั้งของการทำงานที่รหัสต้นฉบับ เพื่อเป็นการสะท้อนเวลาการทำงานของอัลกอริทึม แบ่งการนับได้ 2 วิธีคือ

1. การนับคำสั่งเฉพาะคำสั่งตัวแทน วิธีการนับคำสั่งเฉพาะคำสั่งตัวแทน เป็นการนับคำสั่งบางคำสั่งที่เป็นตัวแทนการทำงานของอัลกอริทึม ที่มีแนวโน้มการเติบโตของเวลาการทำงานของอัลกอริทึม เป็นการนับเพื่อนำผลลัพธ์เป็นตัวแทนของแต่ละอัลกอริทึม ในการสะท้อนอัตราการเติบโตของเวลาการทำงานของแต่ละอัลกอริทึมหรือแต่ละรหัสต้นฉบับ ผู้ที่วิเคราะห์จะต้องระบุคำสั่งตัวแทนที่บรรทัดของรหัสต้นฉบับที่เป็นตัวแทนของอัลกอริทึมนั้นๆให้ถูกต้อง ถ้าระบุคำสั่งตัวแทนไม่ถูกต้องก็จะทำให้การวิเคราะห์แนวโน้มการเติบโตของเวลาการทำงานของอัลกอริทึมนั้นผิดพลาดได้

2. การนับคำสั่งทุกคำสั่ง วิธีการนับคำสั่งทุกคำสั่งของรหัสต้นฉบับในแต่ละอัลกอริทึม ในระหว่างที่โปรแกรมทำงาน เป็นการนับที่สามารถสะท้อนการทำงานของอัลกอริทึมได้ทั้งหมดทุกช่วงการทำงานของรหัสต้นฉบับ เป็นการนับที่สามารถเปรียบเทียบแนวโน้มการเติบโตของเวลาการทำงานในแต่ละช่วงของอัลกอริทึม เพื่อใช้ในการหาช่วงของคำสั่งที่เป็นตัวแทนของอัลกอริทึมที่ใช้ในการวิเคราะห์ประสิทธิภาพเชิงเวลาการทำงานของอัลกอริทึมนั้นๆ ทำให้การวิเคราะห์อัลกอริทึมเป็นไปอย่างถูกต้องมากกว่าการระบุคำสั่งที่เป็นตัวแทนโดยวิเคราะห์

การนับคำสั่งทั้งสองวิธีนั้นอาจจะใช้วิธีการนับทางคณิตศาสตร์ หรือการนับด้วยวิธีการทดลอง การวิเคราะห์อาจเป็นการวิเคราะห์กรณีช้าที่สุด คือใช้เวลาการทำงานมากที่สุด หรือการวิเคราะห์กรณีหาค่าเวลาเฉลี่ยที่ใช้ในแต่ละการทำงานของอัลกอริทึม

2.1.2. **โพรไฟล์เลอร์**

โพรไฟล์เลอร์ [5] คือเครื่องมือที่ช่วยในการเก็บข้อมูลทางด้านเวลาการทำงานของโปรแกรมในขณะที่โปรแกรมทำงาน เพื่อใช้ในการวิเคราะห์พฤติกรรมของโปรแกรม โดยการทำงานของโพรไฟล์เลอร์แบ่งได้เป็น 3 วิธีดังนี้

1. การเก็บข้อมูลตามเหตุการณ์ที่สนใจ (Event-Based) เป็นการเก็บข้อมูลเมื่อเมท็อดที่สนใจทำงานหรือเลิกทำงาน เมื่อจบการทำงานทีละคำสั่งเป็นต้น เป็นวิธีที่ทำให้โปรแกรมทำงานช้ามาก

2. การชักตัวอย่าง (Sampling) เป็นการเก็บข้อมูลเป็นระยะจาก Stack Trace ของโปรแกรม โดยโปรแกรมจะถูกขัดจังหวะเพื่อเก็บข้อมูล แต่ก็ยังสามารถทำงานได้เร็วเต็มที่ แต่ค่าที่เก็บได้เป็นค่าประมาณเชิงสถิติเท่านั้น

3. การแทรกรหัสคำสั่งเพื่อเก็บข้อมูลภายในโปรแกรม (Code Instrumentation) เป็นการแทรกรหัสคำสั่งเพื่อเก็บข้อมูลในขณะโปรแกรมทำงาน โดยผู้เขียนโปรแกรมเป็นผู้แทรกเองหรือให้ระบบแทรกขณะโปรแกรมทำงาน

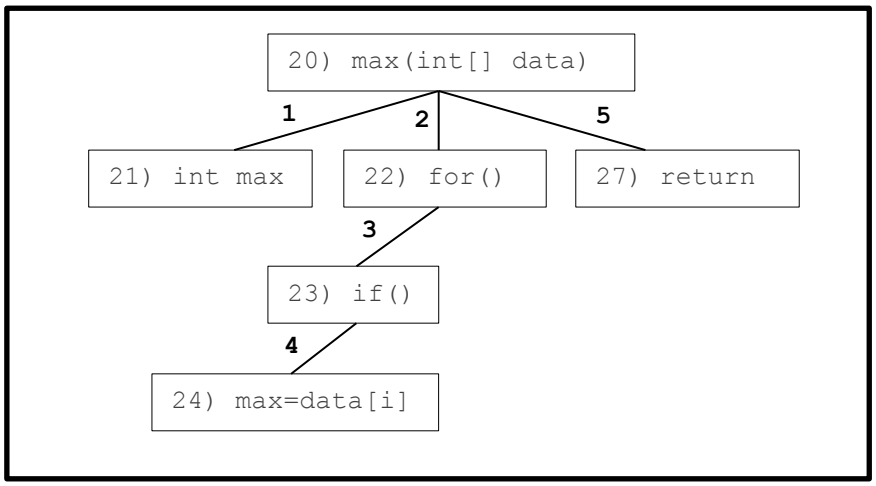
โพรไฟล์เลอร์ที่ใช้ในการเก็บข้อมูลการทำงานของโปรแกรมในขณะที่โปรแกรมทำงานจะอาศัยกลไกที่จาวามีให้คือ JVMPI (Java Profiling Interface) [6] และ JVMTI (JVM Tool Interface) [7] แต่การพัฒนาโพรไฟล์เลอร์ในลักษณะนี้จะขึ้นอยู่กับแต่ละระบบ

2.1.3. **ต้นไม้ไวยากรณ์ที่เป็นนามธรรม** (Abstract Syntax Tree)

AST (Abstract Syntax Tree) [8] เป็นโครงสร้างรูปแบบต้นไม้ที่ใช้เป็นตัวแทนโครงสร้างนามธรรมของรหัสโปรแกรม แต่ละโหนดของต้นไม้หมายถึงโครงสร้างที่เกิดขึ้นในรหัสโปรแกรม

```
20)    public int max(int[] data) {
21)        int max = data[0];
22)        for (int i = 1; i < data.length; i++) {
23)            if (max < data[i]) {
24)                max = data[i];
25)            }
26)        }
27)        return max;
28)    }
```

รหัสที่ 2.1 ตัวอย่างรหัสจาวาที่ใช้อธิบายรูปแบบ AST



ภาพที่ 2.1 รูปแบบต้นไม้ไวยากรณ์แบบนามธรรมจากรหัสตัวอย่างจากรหัสที่ 2.1

ภาพที่ 2.1 แสดงตัวอย่างของโครงสร้างไวยากรณ์แบบนามธรรมของรหัสต้นฉบับ (รหัสที่ 2.1) การอ่านค่าโหนดนั้นจะอ่านค่าตามหมายเลขบรรทัดในรหัสที่ 2.1 โดยเรียงลำดับการทำงานเป็นแบบก่อนลำดับ (Preorder) การทำงานจะเริ่มทำงานจากกิ่งทางซ้ายไปยังกิ่งทางด้านขวาไปจนจบโหนดสุดท้ายของต้นไม้ไวยากรณ์ แต่ละโหนดของต้นไม้ในภาพที่ 2.1 แสดงแทนคำสั่งในแต่ละบรรทัดของรหัสต้นฉบับ

ระบบจะใช้ลักษณะของต้นไม้ไวยากรณ์ที่เป็นนามธรรม(AST) เพื่อช่วยจัดการรหัสต้นฉบับให้อยู่ในรูปแบบ AST และแทรกตัวนับคำสั่งการทำงานลงไปในแต่ละโหนดของ AST เปรียบเสมือนการแทรกคำสั่งลงไปยังทุกๆบรรทัดของรหัสต้นฉบับนั่นเอง

2.2. เอกสารและงานวิจัยที่เกี่ยวข้อง

2.2.1. Algorithm Analysis

Algorithm Analysis a Technical Report [9] บทความนี้นำเสนอการพัฒนาระบบที่ช่วยในการวิเคราะห์อัลกอริทึมเชิงทดลอง การพัฒนานั้นอยู่บนพื้นฐานภาษาจาวา โดยในบทความได้กล่าวว่า การวิเคราะห์อัลกอริทึมแต่เดิมมักจะเน้นในหมู่นักวิทยาศาสตร์คอมพิวเตอร์และจะเน้นในทฤษฎีของการวิเคราะห์อัลกอริทึมในกรณีที่ช้าที่สุด และกรณีค่าเฉลี่ย อย่างไรก็ตาม การวิเคราะห์ทฤษฎีไม่สามารถบอกเรื่องราวทั้งหมดเกี่ยวกับประสิทธิภาพที่แท้จริงของแต่ละอัลกอริทึม จึงต้องมีการวิเคราะห์การเติบโตของผลลัพธ์ที่ได้จากการทดลอง การวิเคราะห์อัลกอริทึมเชิงทดลองเป็นสิ่งที่สามารถบ่งบอกประสิทธิภาพที่แท้จริงของแต่ละอัลกอริทึมได้ และบอกแนวทางในการวิเคราะห์อัลกอริทึมเชิงทดลอง โดยงานวิจัยนี้ได้พัฒนา Timing API ที่ใช้วิเคราะห์อัลกอริทึมเชิงเวลาการทำงานในรูปแบบต่างๆ แต่ยังไม่เห็นภาพการทำงานของแต่ละรหัสในแต่ละอัลกอริทึม

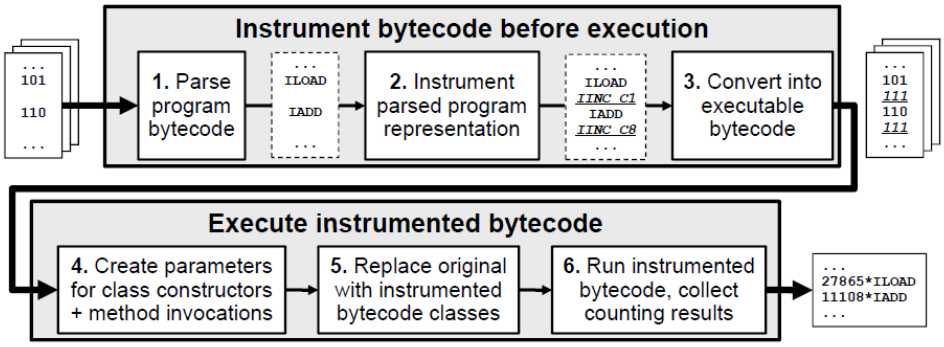
2.2.2. JP

JP [10] เป็นเครื่องมือจากงานวิจัย Exact and Portable Profiling for the JVM Using Bytecode Instruction Counting ที่นำเสนอโพรไฟล์เลอร์เพื่อวัดประสิทธิภาพการทำงานของโปรแกรมภาษาจาวาจากการวัดความถี่การทำงานของรหัสไบต์(Byte Code) ในขณะที่โปรแกรมทำงาน โดยแปลงรหัสต้นฉบับเป็นรหัสใหม่ที่มีคำสั่งนับแทรกอยู่ และจะทำการนับเพิ่มตามจำนวนรหัสดำเนินการที่ใช้ในแต่ละชุดรหัสคำสั่ง เพื่อเปรียบเทียบเวลาการทำงานของแต่ละซีพียูที่แตกต่างกัน เป็นการวัดประสิทธิภาพโดยรวมของโปรแกรมผ่านรหัสดำเนินการ และผลลัพธ์ที่ได้ไม่สามารถระบุบรรทัดหรือช่วงของตัวแทนของรหัสในแต่ละอัลกอริทึมได้

2.2.3. ByCounter

ByCounter [11] เป็นโพรไฟล์เลอร์ที่วัดประสิทธิภาพการทำงานของโปรแกรมภาษาจาวา โดยวัดจากจำนวนครั้งที่คำสั่งในภาษาจาวาเรียกใช้งานในระดับรหัสไบต์ (Byte Code) ในขณะที่โปรแกรมกำลังทำงาน และแสดงผลลัพธ์ในรูปแบบไฟล์อักษรเพื่อแสดงจำนวนครั้งที่โปรแกรมเรียกใช้ในแต่ละรหัสดำเนินการ (Operation Code) ภาพที่ 2.2 แสดงกระบวนการทำงานของการวัดคำสั่งการทำงานด้วยการนับรหัสดำเนินการ โดยเริ่มจากการนำรหัสต้นฉบับแปลงเป็นรหัสไบต์ และแทรกตัวแปรในการนับคำสั่งลงในแต่ละรหัสไบต์ เมื่อโปรแกรมทำงานเสร็จก็จะได้ผลลัพธ์การทำงานของโปรแกรมในมุมมองของแต่ละรหัสดำเนินการ สามารถวัดประสิทธิภาพของอัลกอริทึมแบบภาพรวมทั้งหมด แต่ไม่สามารถตรวจสอบได้ว่าบรรทัดไหนของ

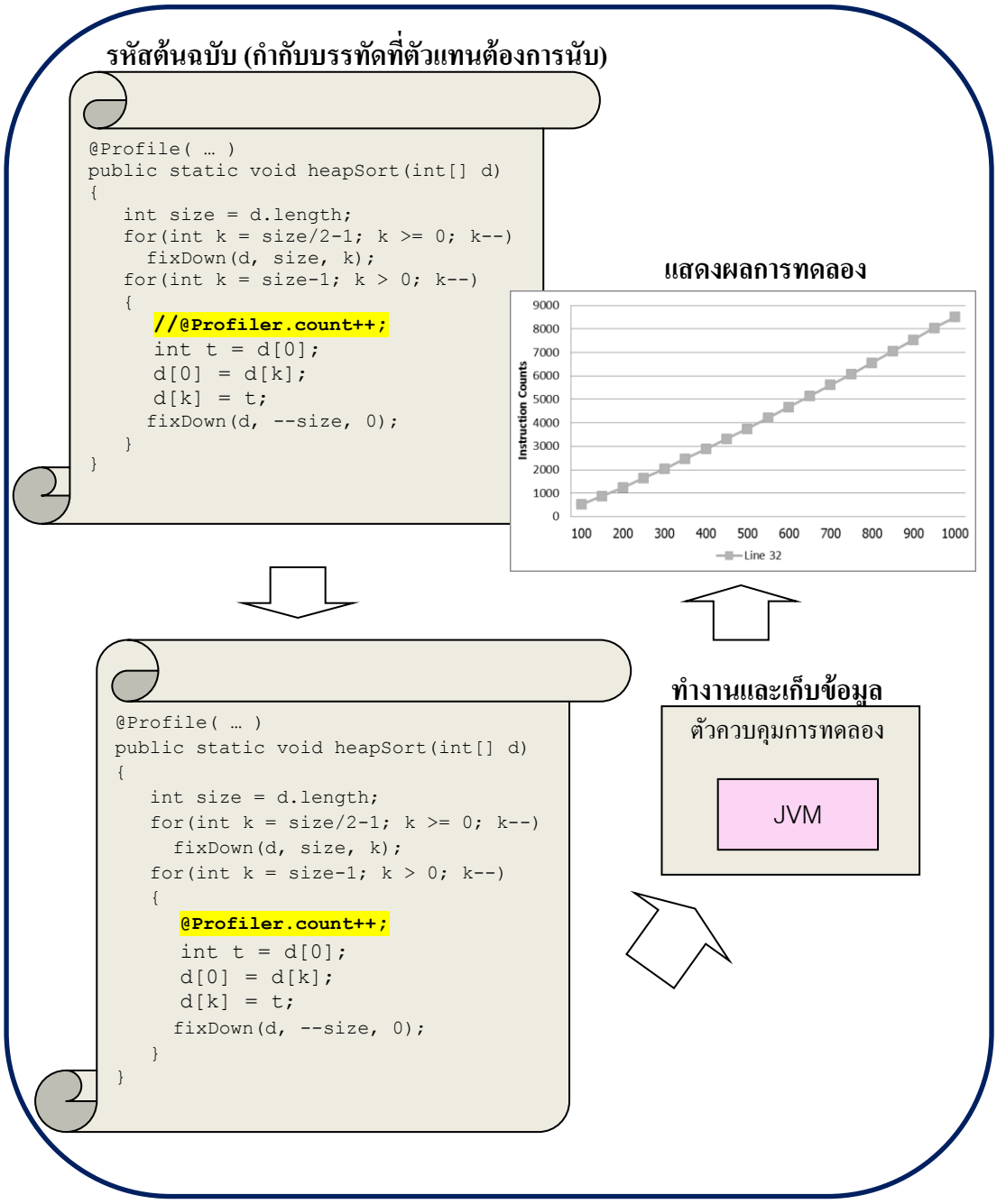
รหัสต้นฉบับเป็นส่วนที่เรียกใช้งานรหัสดำเนินการนั้นๆ ทำให้ไม่เห็นภาพการทำงานของอัลกอริทึมในแต่ละบรรทัดของรหัส



ภาพที่ 2.2 วิธีการวัดและแนวทางการนับคำสั่งด้วย ByCounter

2.2.4. **ตัวควบคุมและคลาสอรรถประโยชน์สำหรับการวิเคราะห์อัลกอริทึมเชิงทดลอง** (JProfile101)

ตัวควบคุมและคลาสอรรถประโยชน์สำหรับการวิเคราะห์อัลกอริทึมเชิงทดลอง (JProfile101) [4] เป็นงานวิจัยที่นำเสนอตัวควบคุมการทดลอง และคลาสอรรถประโยชน์สำหรับการวิเคราะห์อัลกอริทึมเชิงทดลอง เพื่อใช้ในการวิเคราะห์ประสิทธิภาพเชิงเวลาของอัลกอริทึม โดยผู้ทดลองต้องแทรกคำสั่งเก็บข้อมูลที่ต้องการลงในรหัสต้นฉบับด้วยการระบุบรรทัดที่เป็นตัวแทนของอัลกอริทึมที่ต้องการ และกำหนดรูปแบบการทดลองด้วยการกำกับเมท็อดที่สนใจ อันได้แก่ ลักษณะและปริมาณข้อมูลขาเข้า และรูปแบบการแสดงผล โดยวนทำการทดลองตามรูปแบบที่กำหนดอย่างอัตโนมัติ



ภาพที่ 2.3 ภาพรวมของงานวิจัยตัวควบคุมและคลาสอรรถประโยชน์สำหรับการวิเคราะห์อัลกอริทึมเชิงทดลอง

ภาพที่ 2.3 แสดงภาพรวมของการทำงานโดยกำหนดตัวควบคุมการทดลองบนเมท็อดหรือคลาสที่สนใจเพื่อควบคุมข้อมูลขาเข้าโดยใช้การกำกับ Annotation ที่ชื่อ @Profile รวมทั้งกำหนดบรรทัดที่เป็นตัวแทน หรือบรรทัดที่น่าสนใจของอัลกอริทึมในรหัสต้นฉบับด้วยการแทรก “//@Profiler.count++;” ระบบจะทำการนับเฉพาะคำสั่งที่ผู้ใช้งานแทรกนี้เท่านั้น เมื่อเริ่ม

การทดลองระบบจะแปลคำสั่ง “//@Profiler.count++;” ให้กลายเป็น “@Profiler.count++;” จากนั้นจึงแปลงคำสั่งเป็นรหัสไบต์ จากนั้นตัวควบคุมการทดลองจะทำการทดลองและแสดงผลลัพธ์ตามที่ผู้ทดลองกำหนดไว้ การกำหนดบรรทัดที่สนใจ หรือบรรทัดที่เป็นตัวแทนนั้น ผู้ทดลองต้องมีความชำนาญมากพอ จึงสามารถกำหนดบรรทัดตัวแทนของแต่ละรหัสอัลกอริทึมได้อย่างถูกต้อง กรณีที่ผู้ทดลองเป็นผู้ที่ยังไม่ชำนาญมากพออาจก่อให้เกิดความผิดพลาดในการระบุบรรทัด อันเนื่องมาจากประสบการณ์ของผู้ใช้งาน ดังนั้นผลลัพธ์ที่ได้จากการทดลองอาจจะไม่ใช่ผลลัพธ์ที่ใช้ในการวิเคราะห์ประสิทธิภาพเชิงเวลาที่ดีที่สุด

บทที่ 3

วิธีดำเนินการวิจัย

งานวิจัยนี้นำเสนอเครื่องมือเพื่อช่วยในการวิเคราะห์อัลกอริทึมเชิงทดลองด้วยการวัดคำสั่งการทำงาน ทำการทดลองและแสดงผลลัพธ์ที่สะท้อนอัตราการเติบโตของเวลาการทำงานของอัลกอริทึมอย่างอัตโนมัติ โดยมีวิธีดำเนินการวิจัยดังต่อไปนี้

1. ค้นคว้าและศึกษาทฤษฎีและงานวิจัยที่เกี่ยวข้อง เพื่อนำทฤษฎีที่ศึกษาและค้นคว้ามาประยุกต์ใช้ในงานวิจัยเพื่อให้ได้ผลลัพธ์ตามวัตถุประสงค์ของงานวิจัย โดยทฤษฎีและงานวิจัยที่สำคัญได้แก่ ทฤษฎีต้นไม้ไวยากรณ์ที่เป็นนามธรรม (Abstract Syntax Tree) [8], การแทรกรหัสเพื่อวัดคำสั่งการทำงาน, รวมทั้งงานวิจัย ตัวควบคุมและคลาสอรรถประโยชน์สำหรับการวิเคราะห์อัลกอริทึมเชิงทดลอง เป็นต้น

2. พัฒนาเครื่องมือเพื่อใช้ทดสอบแนวความคิด โดยนำทฤษฎีและงานวิจัยที่เกี่ยวข้องมาพัฒนาและทดสอบ เพื่อหาความเป็นไปได้ในการนำทฤษฎีต่างๆที่ได้ค้นคว้าและศึกษาเพื่อนำมาใช้ในงานวิจัยตามวัตถุประสงค์ของงานวิจัย

3. นำทฤษฎีและงานวิจัยที่เกี่ยวข้องที่เป็นผลลัพธ์จากการศึกษาและทดลอง มาพัฒนาประยุกต์เพื่อเป็นเครื่องมือเพื่อช่วยในการวิเคราะห์อัลกอริทึมเชิงทดลองด้วยการวัดคำสั่งการทำงาน

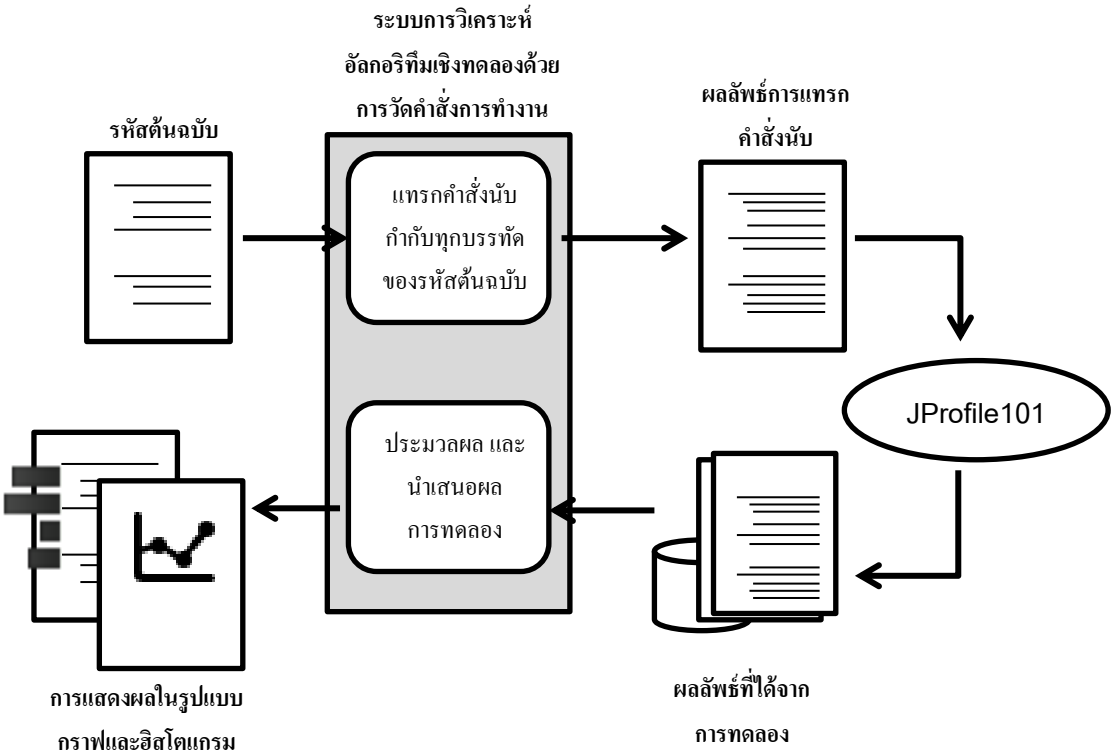
4. ทดสอบและแก้ไขเครื่องมือโดยทำการทดลองกับอัลกอริทึมที่หลากหลาย และรูปแบบของรหัสต้นฉบับที่หลากหลาย เพื่อใช้ในการหาข้อบกพร่องของเครื่องมือ ทำการปรับปรุงและทดสอบซ้ำ

5. ปรับปรุงรูปแบบการนำเสนอผลการทดลองของแต่ละอัลกอริทึม โดยงานวิจัยเลือกการนำเสนอในรูปแบบกราฟกราฟเส้น เพื่อสะท้อนอัตราการเติบโตของเวลาการทำงานของอัลกอริทึมได้อย่างชัดเจน รวมทั้งการนำเสนอในรูปแบบกราฟแท่งฮิสโตแกรม ในแต่ละบรรทัดของรหัสต้นฉบับ เพื่อสะท้อนให้เห็นการทำงานของแต่ละบรรทัดของรหัสต้นฉบับได้อย่างชัดเจน ทำให้เห็นภาพและทำความเข้าใจในการทำงานของแต่ละอัลกอริทึมได้ชัดเจนมากยิ่งขึ้น

ระบบที่พัฒนาขึ้นนี้ให้บริการการวิเคราะห์อัลกอริทึมเชิงทดลองด้วยการทำงานผ่านบริการเว็บ โดยผู้ทำการทดลองสามารถทดลองและดูผลลัพธ์การทดลองทางหน้าเว็บ รวมทั้งสามารถเขียนโปรแกรมเพื่อเรียกใช้เครื่องมือผ่านรูปแบบเว็บเซอร์วิส

3.1. ภาพรวมและขั้นตอนการทำงานของระบบ

เริ่มต้นจากการนำ JProfile101 [4] (ตัวควบคุมและคลาสอรรถประโยชน์สำหรับการวิเคราะห์อัลกอริทึมเชิงทดลอง) มาใช้กับงานวิจัย โดย JProfile101 มีความสามารถในการควบคุมการทดลอง และทำการทดลองซ้ำอย่างอัตโนมัติ โดยผู้ใช้งานจะต้องระบุบรรทัดที่เป็นตัวแทนของอัลกอริทึม เพื่อแทรกรหัส “//@Profiler.counts[ ]++;” ในการนับลงในรหัสต้นฉบับและกำกับการทดลองด้วย @Profile ดังนั้นการทำงานของเครื่องมือเพื่อช่วยในการวิเคราะห์อัลกอริทึมเชิงทดลองด้วยการวัดคำสั่งการทำงานต้องทำหน้าที่ในการกำหนดข้อมูลตั้งต้นต่างๆ รวมทั้งแทรกรหัสในการนับกำกับลงในรหัสต้นฉบับทุกบรรทัด ก่อนที่จะส่งต่อไปยัง JProfile101 และนำผลลัพธ์ที่ได้จากการทดลองมาใช้ในการประมวลผล และแสดงผลลัพธ์ในรูปแบบต่างๆ ต่อไป ดังภาพที่ 3.1

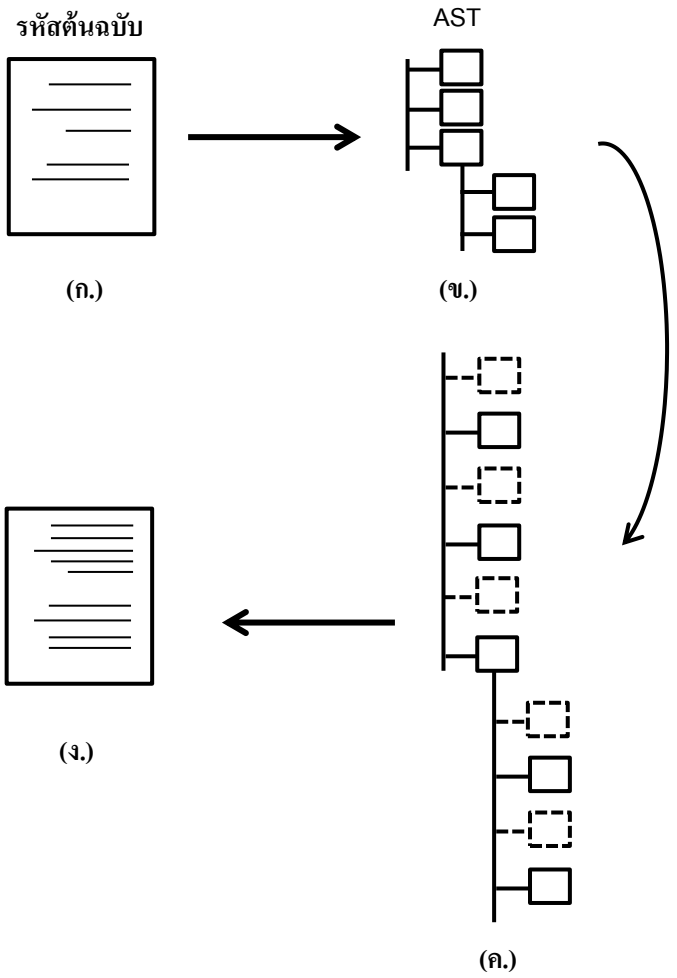


ภาพที่ 3.1 ภาพรวมการทำงานของระบบการวิเคราะห์อัลกอริทึมเชิงทดลองด้วยการวัดคำสั่งการทำงาน

ภาพที่ 3.1 จะเห็นว่าระบบการวิเคราะห์อัลกอริทึมเชิงทดลองด้วยการวัดคำสั่ง การทำงานจะแบ่งการทำงานออกเป็น 3 ขั้นตอน คือ การแทรกคำสั่งนับทุกคำสั่ง, การทดลองผ่าน JProfile101 และการประมวลผลข้อมูลที่ได้จากการทดลองพร้อมทั้งแสดงผล

3.1.1. การแทรกคำสั่งนับทุกคำสั่ง

การแทรกคำสั่งนับลงในรหัสต้นฉบับทุกคำสั่งนั้น อาศัยทฤษฎีต้นไม้ไวยากรณ์ที่เป็นนามธรรม หรือ AST (Abstract Syntax Tree) [8] ในการแปลงรหัสต้นฉบับให้เป็นรูปแบบของโหนดของต้นไม้ และระบบจะทำการแทรกโหนดคำสั่งในการนับลงไปในแต่ละโหนดของรหัสต้นฉบับอย่างอัตโนมัติ



ภาพที่ 3.2 แสดงการแปลงรหัสต้นฉบับเป็น AST

ภาพที่ 3.2 เราจะแปลงรหัสต้นฉบับจากรูปแบบของ Text 3.2 (ก.) ไปยัง AST 3.2 (ข.) ที่มีลักษณะเป็นรูปโหนดของต้นไม้ จากนั้น 3.2 (ค.) แสดงให้เห็นการแทรกโหนดคำสั่งในการ

นับเข้าไปใน AST โดยมองคำสั่งนับนั้นเป็นอีกโหนดหนึ่งของต้นไม้ หลังจากการแทรกครบทุกโหนดของต้นไม้แล้ว ก็จะแปลงกลับเป็นรหัสใหม่ที่มีคำสั่งนับกำกับทุกบรรทัด 3.2 (ง.)

คำสั่งนับที่กำกับอยู่ในแต่ละบรรทัดของรหัส จะเป็นลักษณะตามรูปแบบของ JProfile101 เนื่องจากภายหลังการแทรกรหัสคำสั่งนับแล้วจะนำรหัสผลลัพธ์ที่ได้ไปยัง JProfile101 เพื่อทำการทดลองตามรูปแบบที่กำหนดอย่างอัตโนมัติ โดยที่การแทรกโหนดของคำสั่งนับอาศัยลักษณะของ AST โดยมีวิธีการแทรกคำสั่งดังนี้

1. อ่านข้อมูลรหัสจาวาที่เป็นรูปแบบอักษรเข้ามาในระบบ และทำการแปลงข้อมูลตัวอักษรให้อยู่ในรูปแบบ AST โดยใช้เครื่องมือของ Eclipse IDE [12] ที่มีการเตรียมคลาส และเมธ็อดต่างๆที่สามารถจัดการรหัสในรูปแบบ AST ได้ การแปลงรหัสจาวาที่เป็นรูปแบบอักษรให้อยู่ในรูปแบบของ AST นั้นสามารถทำได้โดยใช้งานคลาส org.eclipse.jdt.core.dom.ASTParser สามารถสร้างรูปแบบ AST จากรหัสจาวาที่ได้รับมา ตัวอย่างรหัสในการแปลงรหัสจาวาเป็น AST ดังภาพที่ 3.3

```
ASTParser parser = ASTParser.newParser(AST.JLS3);
parser.setKind(ASTParser.K_COMPILATION_UNIT);
Document doc = new Document(javaText);
parser.setSource(doc.get().toCharArray()); // set source
CompilationUnit node = (CompilationUnit) parser.createAST(null); // parse
AST ast = node.getAST();
```

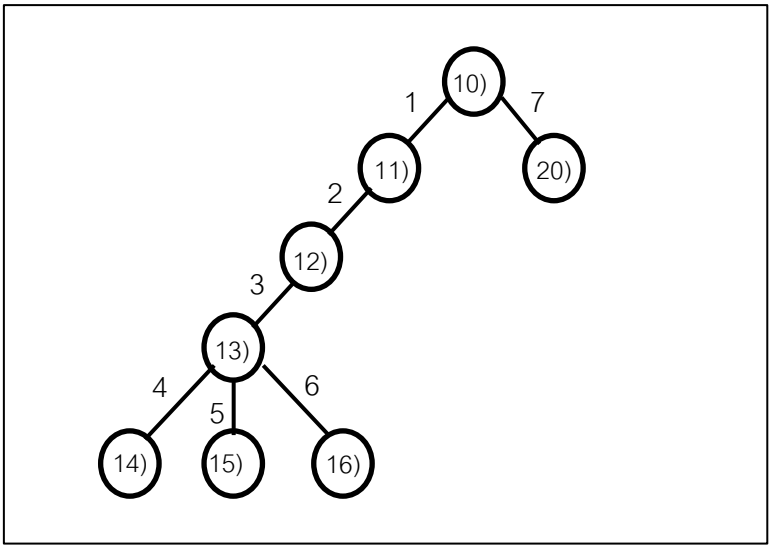
ภาพที่ 3.3 ตัวอย่างรหัสในการแปลงรหัสจาวาให้อยู่ในรูปแบบ AST

2. จากนั้นจึงสร้างตัวแปรจาก org.eclipse.jdt.core.dom.rewrite.ASTRewrite ที่เป็นคลาสในการจัดการ AST เพื่อปรับปรุงแก้ไขโหนดต่างๆของ AST ที่สร้างขึ้นจากข้อที่ 1

3. การสำรวจแต่ละโหนดของ AST เป็นการสำรวจแบบก่อนลำดับ (Preorder)

```
10) public int[] bubbleSort(int[] data) {
11)    for (int i = 0; i < data.length; i++) {
12)        for (int j = i; j < data.length; j++) {
13)            if (data[i] > data[j]) {
14)                int temp = data[i];
15)                data[i] = data[j];
16)                data[j] = temp;
17)            }
18)        }
19)    }
20)    return data;
21) }
```

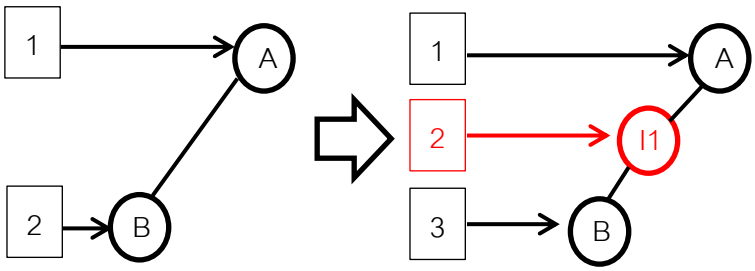
รหัสที่ 3.1 ตัวอย่างรหัส Bubble Sort



ภาพที่ 3.4 ตัวอย่างลำดับการสำรวจโหนดต้นไม้ของรหัสจาวา

ภาพที่ 3.4 แสดงรูปแบบการสำรวจโหนดของต้นไม้แบบก่อนลำดับ โดยไล่สำรวจตามหมายเลขเส้นเชื่อมของโหนด เริ่มจากหมายเลข 1 ถึงหมายเลข 7 โดยการสำรวจในแต่ละโหนดนั้น สามารถสำรวจได้ถึงระดับของ Expression ในแต่ละ Statement ของรหัสจาวา แต่ในงานวิจัยนี้ จะมองโหนดของต้นไม้เฉพาะระดับบรรทัดการทำงานของรหัสจาวา ตัวเลขที่แสดงภายในโหนดหมายถึงเลขบรรทัดของรหัสจาวา bubbleSort(int[] data) ที่แสดงในรหัสที่ 3.1 นั่นเอง

การสำรวจต้นไม้แบบก่อนลำดับทำให้การแทรกโหนดคำสั่งนับไม่สามารถทำได้เลยทันทีเมื่อการสำรวจเข้าถึงเข้าไปถึงโหนดต่างๆ เนื่องจากการแทรกโหนดคำสั่งนับจะทำให้เกิดโหนดใหม่ขึ้นทันที จึงทำให้การสำรวจต้นไม้ไม่มีวันจบ



(ก.) (ข.)

ภาพที่ 3.5 ภาพตำแหน่งหลังจากการแทรกโหนดคำสั่งนับ

ภาพที่ 3.5 แสดงตำแหน่งของโหนดต้นไม้ภายหลังการแทรกคำสั่งนับ จากภาพที่ 3.5 (ก.) ตำแหน่งของโหนด A คือ 1 และตำแหน่งของโหนด B คือ 2 เมื่อสำรวจถึงโหนด A ตำแหน่งการเข้าสำรวจอยู่ที่ตำแหน่งลำดับที่ 1 ต่อมาเมื่อสำรวจถึงโหนด B ตำแหน่งการเข้าสำรวจจะอยู่ตำแหน่งลำดับที่ 2 เมื่อแทรกโหนดคำสั่งนับ จากภาพที่ 3.5 (ข.) มีการแทรกโหนด I1 ระหว่างโหนด A และโหนด B ดังนั้นตำแหน่งของโหนด I1 จึงเป็นโหนดลำดับที่ 2 ซึ่งเท่ากับเปลี่ยนแปลงลำดับของโหนด B ให้ไปอยู่ลำดับที่ 3 เมื่อพบว่ามีโหนดลำดับต่อไป การสำรวจจึงยังไม่สิ้นสุด เมื่อสำรวจต่อไปที่โหนดลำดับที่ 3 ซึ่งเป็นโหนดที่ต้องแทรกคำสั่งนับ เมื่อแทรกคำสั่งนับก็เท่ากับว่าโหนด B ถูกเปลี่ยนลำดับต่อไปอย่างไม่สิ้นสุด

```
if (condition)
    statement;
```

```
if (condition)
{
    statement;
}
```

(ก.) (ข.)

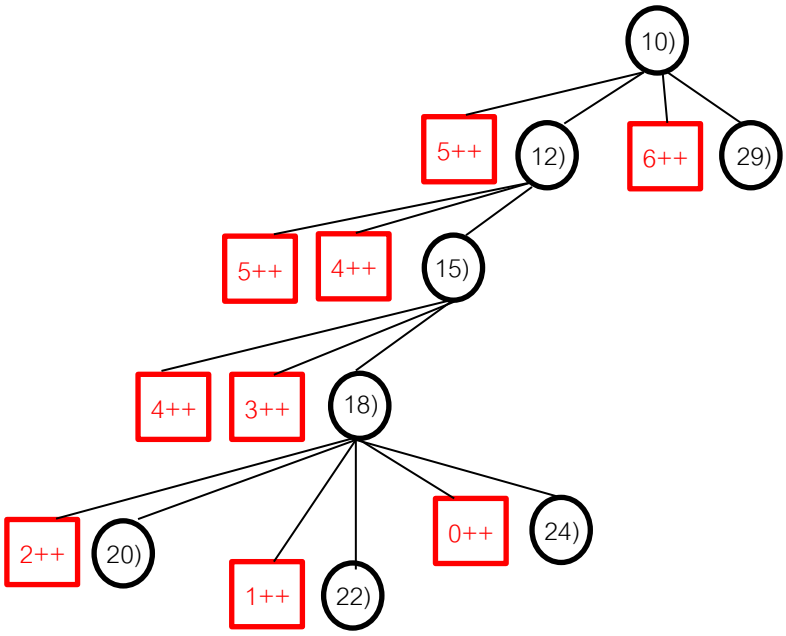
ภาพที่ 3.6 ตัวอย่างรหัสที่ไม่มีบล็อคและรหัสภายหลังการสร้างบล็อค

การแทรกโหนดคำสั่งนับ มีผลกระทบต่อรหัสจาวาต้นฉบับทำให้บรรทัดของรหัสจาวาเกิดการเปลี่ยนแปลง ซึ่งต้องเก็บรวบรวมข้อมูลของรหัสต้นฉบับระหว่างการทดลอง เช่น ลำดับของแต่ละบรรทัดของรหัสต้นฉบับ, จุดเริ่มต้นและสิ้นสุดของบล็อคคำสั่ง และตำแหน่งของบรรทัดในบล็อคคำสั่ง ดังนั้นการสำรวจโหนดต้นไม้จึงต้องเข้าสำรวจ 2 รอบ รอบแรกเป็นการสำรวจเพื่อเก็บข้อมูลต่างๆของแต่ละโหนด เพื่อนำข้อมูลที่ได้ไปใช้ในการบ่งชี้การทำงานที่เกิดขึ้นในแต่ละบรรทัดต่อไป จากนั้นสำรวจรอบที่ 2 เพื่อจัดเก็บโหนดของต้นไม้ลงในอาเรย์ การจัดเก็บนั้นจะต้องอยู่ภายในบล็อคคำสั่ง จากภาพที่ 3.6 (ก.) เป็นตัวอย่างคำสั่งที่ไม่มีบล็อค เมื่อสำรวจลงไปถึงตำแหน่งที่ statement; จะไม่สามารถแทรกโหนดคำสั่งนับได้ เนื่องจากรากของโหนด

statement; นั้นไม่อยู่ภายใต้บล็อคคำสั่ง ดังนั้นการสำรวจในรอบที่ 2 จะสร้างบล็อคของคำสั่งในกรณีที่พบว่าโหนดที่สำรวจนั้นไม่อยู่ภายใต้บล็อค ดังภาพที่ 3.6 (ข.) ระบบจะสร้างบล็อค { statement; } ขึ้นมาครอบโหนดคำสั่งนั้นๆ จากนั้นจะเก็บบล็อคคำสั่งไปยังบล็อคอาเรย์ที่สร้างขึ้นเพื่อใช้แทรกโหนดคำสั่งนับต่อไป

```
10)   public int[] bubbleSort(int[] data) {
11)               //@Profiler.counts[5]++;
12)               for (int i = 0; i < data.length; i++) {
13)                     //@Profiler.counts[5]++;
14)                     //@Profiler.counts[4]++;
15)                     for (int j = i; j < data.length; j++) {
16)                           //@Profiler.counts[4]++;
17)                           //@Profiler.counts[3]++;
18)                           if (data[i] > data[j]) {
19)                                 //@Profiler.counts[2]++;
20)                                 int temp = data[i];
21)                                 //@Profiler.counts[1]++;
22)                                 data[i] = data[j];
23)                                 //@Profiler.counts[0]++;
24)                                 data[j] = temp;
25)                           }
26)                     }
27)               }
28)               //@Profiler.counts[6]++;
29)               return data;
30)         }
```

รหัสที่ 3.2 ตัวอย่างรหัส Bubble Sort ภายหลังการแทรกคำสั่งนับ



ภาพที่ 3.7 ภาพต้นไม้ภายหลังการแทรกคำสั่งนับ

หลังจากการจัดเก็บบล็อคต่างๆที่สำรวจจนครบทุกโหนดของต้นไม้ ให้นำบล็อคอาเรย์ที่ได้จากการสำรวจนั้นมาวนลูปย้อนหลัง โดยนำแต่ละบล็อคที่ได้มาทำงานผ่าน ASTRewrite เพื่อแทรกโหนดคำสั่งนับ และเมื่อการแทรกโหนดคำสั่งเริ่มต้นแทรกจากโหนดด้านหลังของต้นไม้ จากการอ่านบล็อคอาเรย์ย้อนหลัง ไปยังโหนดด้านหน้า จึงไม่ทำให้เกิดการเลื่อนของโหนดที่เป็นสาเหตุของการทำงานแบบไม่รู้จบ ดังภาพที่ 3.7 แสดงการแทรกคำสั่งนับจากบล็อคด้านในสุดซึ่งเป็นบล็อคคำสั่ง if เริ่มต้นบรรทัดที่ 18 สิ้นสุดบรรทัดที่ 25 ของรหัสที่ 3.2 โดยแทรกคำสั่งตามดัชนีลำดับบรรทัดของบล็อค if ที่เก็บไว้จากการสำรวจ และแทรกคำสั่งนับในบล็อคลำดับถัดมาจนถึงบล็อคลำดับแรกสุดคือบล็อคเริ่มต้นบรรทัดที่ 10 สิ้นสุดบรรทัดที่ 30 ของรหัสที่ 3.2 เมื่อแทรกโหนดคำสั่งนับลงในทุกคำสั่งของรหัสจาวาต้นฉบับแล้ว ASTRewrite จะทำหน้าที่เขียนกลับไปยังรูปแบบข้อความโดยแปลงโหนดคำสั่งนับที่เพิ่มขึ้นมาบนต้นไม้ไวยากรณ์ที่เป็นนามธรรมของรหัสจาวากลับเป็นรหัสต้นฉบับได้อย่างถูกต้อง จากนั้นบันทึกผลลัพธ์ของรหัสจาวาที่ได้เพื่อนำไปใช้ในขั้นตอนการทดลองต่อไป

3.1.2. การทดลองผ่าน JProfile101

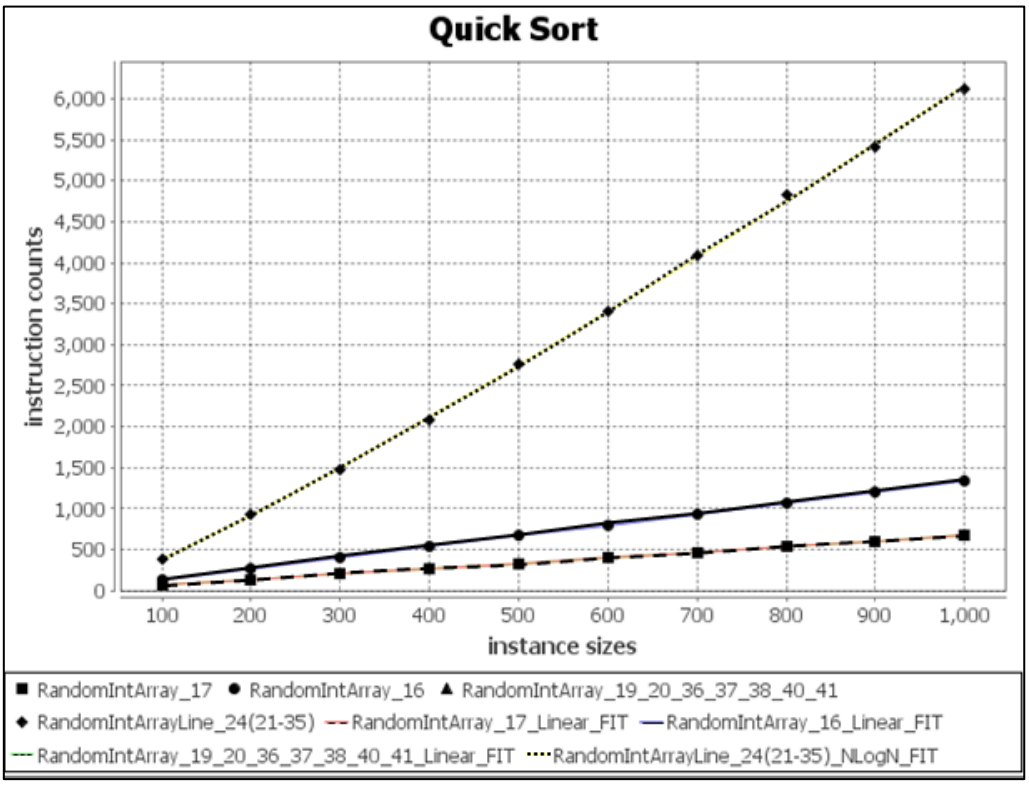
การทดลองผ่าน JProfile101 เป็นการทำงานขั้นตอนต่อจากการแทรกคำสั่งนับลงในรหัสต้นฉบับ โดยนำผลลัพธ์รหัสจาวาที่ได้แทรกคำสั่งนับในข้อที่ 3.1.1 มาทดลองตามรูปแบบ

ของ JProfile101 ด้วยการกำกับรูปแบบการทดลองอันได้แก่ รูปแบบข้อมูลขาเข้า และ รูปแบบการทดลองโดยการเขียน @Profile กำกับไว้ที่เมธ็อดที่ใช้ทดลอง จากนั้นจึงนำข้อมูลผลลัพธ์การทำงานที่ได้ในแต่ละบรรทัดไปใช้ในขั้นตอนการประมวลผลเพื่อการแสดงผลลัพธ์ที่ต้องการต่อไป

3.1.3. การประมวลผลข้อมูลที่ได้จากการทดลองพร้อมทั้งแสดงผล

```
11)     public void quicksort(int[] a) {
12)         quicksort(a, 0, a.length - 1);
13)     }
14)
15)     public void quicksort(int[] a, int left, int right) {
16)         if (right <= left) {
17)             return;
18)         }
19)         int i = left - 1;
20)         int j = right;
21)         while (true) {
22)             while ((a[++i]) < (a[right]))
23)             {}
24)             while ((a[right]) < (a[--j])) {
25)                 if (j == left) {
26)                     break;
27)                 }
28)             }
29)             if (i >= j) {
30)                 break;
31)             }
32)             int swap = a[i];
33)             a[i] = a[j];
34)             a[j] = swap;
35)         }
36)         int swap = a[i];
37)         a[i] = a[right];
38)         a[right] = swap;
39)
40)         quicksort(a, left, i - 1);
41)         quicksort(a, i + 1, right);
42)     }
```

รหัสที่ 3.3 ตัวอย่างรหัส Quicksort



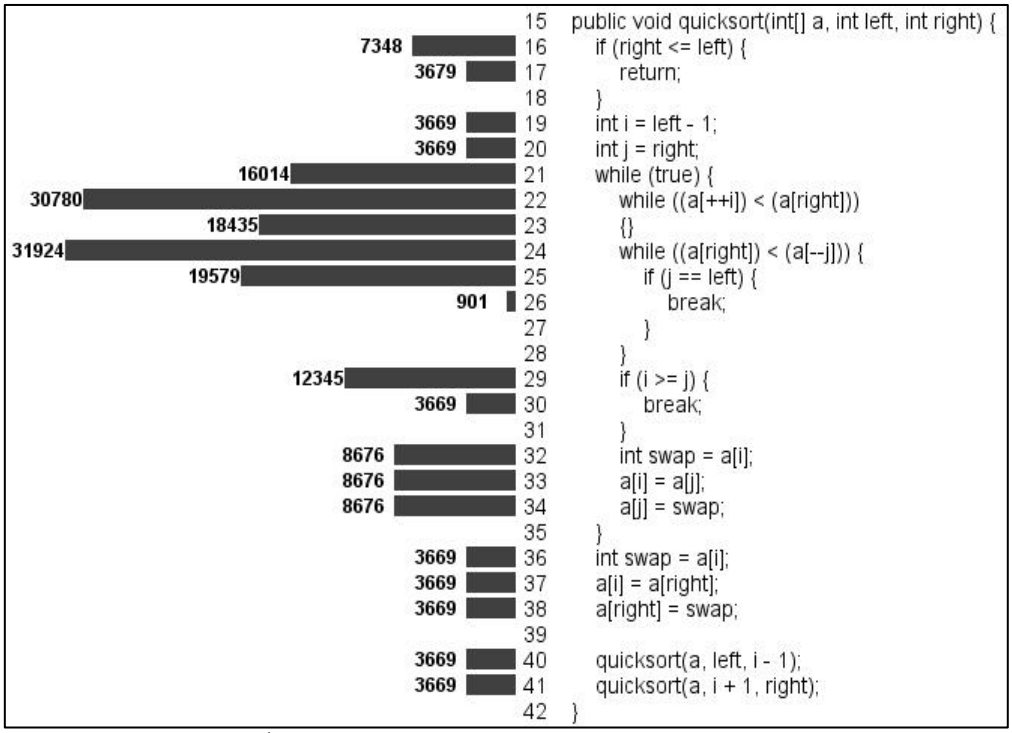
ภาพที่ 3.8 ตัวอย่างกราฟเส้นภายหลังการทดลอง

การประมวลผลข้อมูลและแสดงผลลัพธ์เป็นขั้นตอนการทำงานหลังจากได้รับผลการทดลองจาก JProfile101 โดยเมื่อสิ้นสุดการทดลองระบบจะรับข้อมูลปริมาณการทำงานของแต่ละบรรทัดของรหัสต้นฉบับ โดยนำข้อมูลที่ได้มาจัดกลุ่มข้อมูลเพื่อแสดงผลในรูปแบบกราฟเส้นและแสดงผลในรูปแบบกราฟฮิสโตแกรมดังตัวอย่างในภาพที่ 3.8 และ 3.9 ตามลำดับ โดยจัดกลุ่มข้อมูลได้ดังนี้

1. กลุ่มของช่วงบรรทัดที่เป็นลูป (for, while, ฯลฯ) เลือกแสดงผลเฉพาะบรรทัดตัวแทนของลูปที่ทำงานมากที่สุด ดังภาพที่ 3.8 แสดงผลลัพธ์ของรหัสที่ 3.3 โดยรวมกลุ่มของบรรทัดที่เป็นลูป และแสดงผลลัพธ์เฉพาะบรรทัดตัวแทนของลูปนั้นๆ ตัวอย่างเช่น RandomIntArray_24(21-35) ตัวเลขด้านหลังเครื่องหมาย “_” หมายถึงผลลัพธ์ที่แสดงเป็นของบรรทัดที่ 24 ซึ่งเป็นตัวแทนของรหัสลูปช่วงบรรทัดที่ 21 ถึงบรรทัดที่ 35

2. กลุ่มของบรรทัดของรหัสต้นฉบับที่มีจำนวนครั้งในการทำงานที่เท่ากัน ดังภาพที่ 3.8 แสดงผลลัพธ์ของรหัสที่ 3.3 โดยรวมกลุ่มของบรรทัดที่มีจำนวนครั้งการทำงานที่เท่ากัน และแสดงผลชื่อข้อมูลตัวอย่างเช่น RandomIntArray_19_20_36_37_38_40_41 ตัวเลขที่แสดงหมายถึงหมายเลขบรรทัดที่มีจำนวนครั้งการทำงานที่เท่ากัน

นอกจากการแสดงผลการทดลองแบบกราฟเส้นแล้ว ระบบจะสร้างการแสดงผลการทดลองในรูปแบบกราฟฮิสโตแกรม เพื่อสะท้อนจำนวนครั้งของการทำงานในแต่ละบรรทัดของรหัสต้นฉบับ



ภาพที่ 3.9 ตัวอย่างการแสดงผลกราฟแบบฮิสโตแกรม

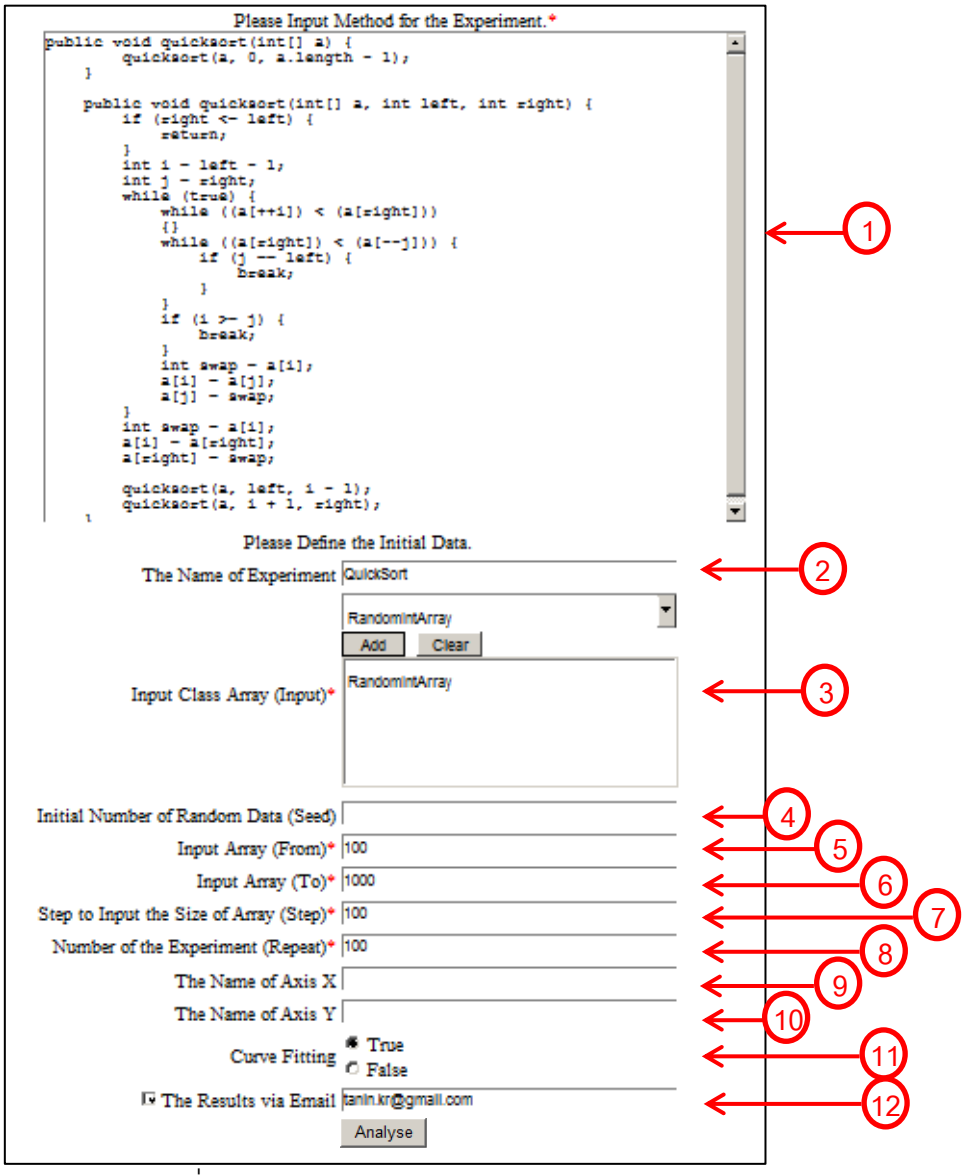
ภาพที่ 3.9 เป็นตัวอย่างฮิสโตแกรม ที่แสดงเป็นกราฟแท่งแนวขวาง แสดงผลการทำงานในแต่ละบรรทัดของรหัสต้นฉบับ โดยสัมพันธ์กับกราฟเส้นในภาพที่ 3.8 เช่น บรรทัดที่เป็นตัวแทนรหัส while loop (RandomIntArrayLine_24(21-35)) เริ่มตั้งแต่บรรทัดที่ 21 ถึงบรรทัดที่ 35 มีตัวแทนคือบรรทัดที่ 24 ที่ทำงานบ่อยครั้งที่สุดของรหัสลูป ซึ่งผลลัพธ์ของกราฟฮิสโตแกรมของข้อมูลสัมพันธ์กันกับกราฟเส้น เพื่อง่ายต่อการทำความเข้าใจในการทำงานของรหัสคำสั่งในแต่ละรหัสอัลกอริทึมมากยิ่งขึ้น

3.2. การพัฒนาการเข้าใช้งานระบบ

การใช้งานระบบการวิเคราะห์อัลกอริทึมเชิงทดลองด้วยการวัดคำสั่ง ได้รับการพัฒนาให้สั่งงานผ่านเว็บ ผู้ใช้งานต้องกำหนดข้อมูลตั้งต้นต่างๆ จากนั้นระบบจะรวบรวมข้อมูลที่

ได้รับจากหน้าเว็บ ส่งต่อไปยังเว็บเซอร์วิสโดยที่เว็บเซอร์วิสจะเรียกใช้งาน JProfile101 ด้วยการส่งค่าพารามิเตอร์ที่สำคัญ จากนั้นโปรแกรมจะตรวจสอบผลลัพธ์ที่ได้จาก JProfile101 และแสดงผลลัพธ์ผ่านหน้าเว็บ โดยเว็บที่ใช้แสดงผลต่างๆ พัฒนาด้วย ASP.Net และเว็บเซอร์วิสในรูปแบบ REST เซอร์วิส

3.2.1. การใช้งานระบบผ่านหน้าเว็บ



ภาพที่ 3.10 ตัวอย่างหน้าจอภาพของเว็บสำหรับทำการทดลอง

ภาพที่ 3.10 เป็นตัวอย่างหน้าเว็บที่พัฒนาขึ้นเพื่อให้ผู้ใช้งานสามารถทดลองอัลกอริทึมมีการรับข้อมูลดังต่อไปนี้

1. รหัสเมธ็อดของอัลกอริทึมที่ใช้ในการทดลอง

2. ชื่อของการทดลอง

3. คลาสของหน่วยผลิตข้อมูลขาเข้าดังตารางที่ 3.1

ชื่อหน่วยผลิตข้อมูล	รายการคลาสหน่วยผลิตข้อมูล
RandomIntArray	ผลิตข้อมูลในอาเรย์แบบ Integer โดยมีค่าแบบสุ่ม
SortedIntArray	ผลิตข้อมูลในอาเรย์แบบ Integer โดยมีค่าเรียงลำดับจากน้อยไปหามาก
ReverseIntArray	ผลิตข้อมูลในอาเรย์แบบ Integer โดยมีค่าเรียงลำดับจากมากไปหาน้อย
RandomDoubleArray	ผลิตข้อมูลในอาเรย์แบบ Double โดยมีค่าแบบสุ่ม
SortedDoubleArray	ผลิตข้อมูลในอาเรย์แบบ Double โดยมีค่าเรียงลำดับจากน้อยไปหามาก
ReverseDoubleArray	ผลิตข้อมูลในอาเรย์แบบ Double โดยมีค่าเรียงลำดับจากมากไปหาน้อย

ตารางที่ 3.1 รายการคลาสหน่วยผลิตข้อมูล

4. ค่าเริ่มต้นในการสุ่มข้อมูลขาเข้า

5. ขนาดเริ่มต้นของข้อมูลขาเข้า

6. ขนาดที่มากที่สุดของข้อมูลขาเข้า

7. ขนาดที่เพิ่มขึ้นในขณะที่โปรแกรมทำงาน โดยเริ่มจากขนาดเริ่มต้น ไปจนถึงขนาดที่มากที่สุดของข้อมูลขาเข้าที่กำหนดไว้ในข้อที่ 5 และข้อที่ 6

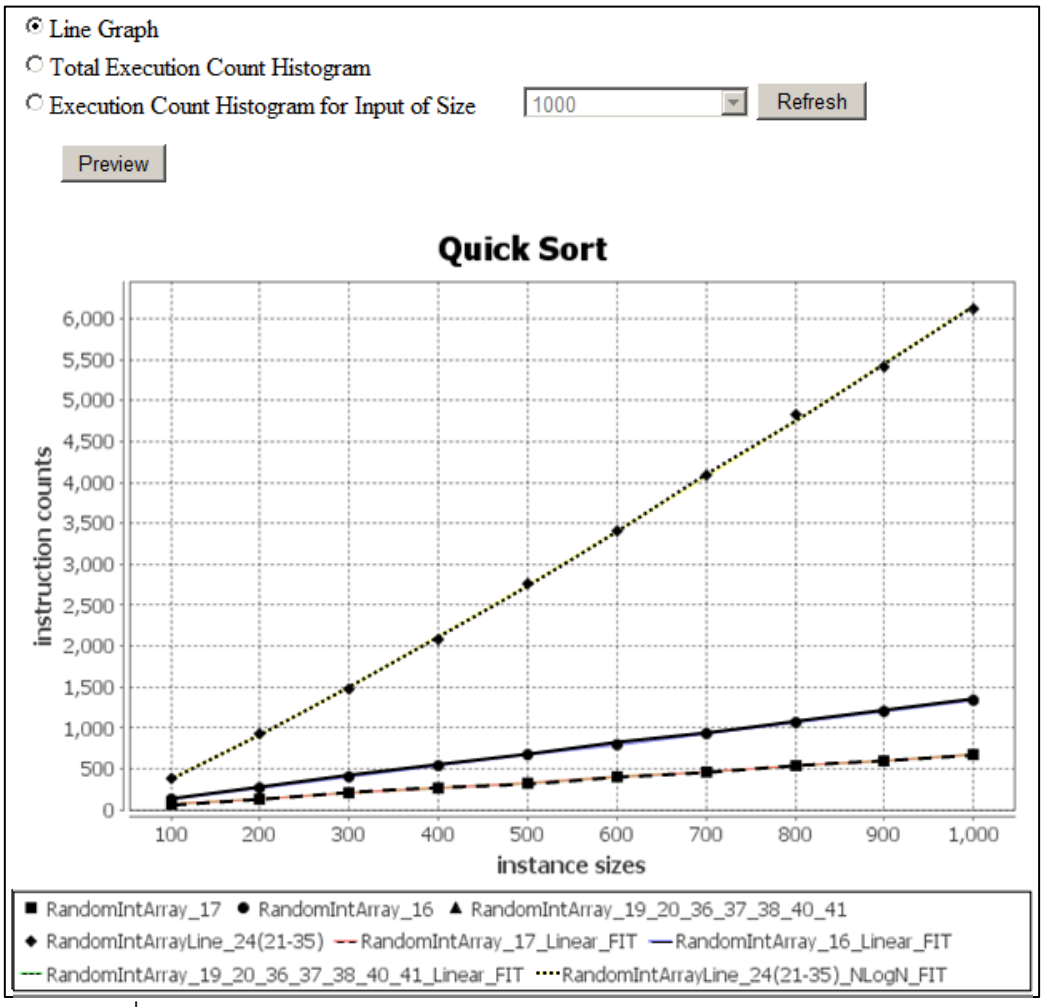
8. จำนวนรอบการทำการทดลองซ้ำ

9. ข้อความกำกับแกน X

10. ข้อความกำกับแกน Y

11. ตัวเลือกการปรับเส้นโค้ง

12. เลือกส่งผลลัพธ์ผ่านทางอีเมลโดยใส่ที่อยู่ของอีเมล



ภาพที่ 3.11 ตัวอย่างผลลัพธ์หลังจากการทดลองแสดงรูปแบบกราฟผ่านหน้าเว็บ

เมื่อกำหนดข้อมูลตั้งต้นทั้งหมดจากภาพที่ 3.10 ครบถ้วนแล้ว ผู้ใช้สามารถสั่งเริ่มการทดลองโดยการกดปุ่ม Analyse ระบบจะทดลองตามรูปแบบการทดลองที่กำหนดไว้อย่างอัตโนมัติ เมื่อการทดลองเสร็จสิ้นจะปรากฏหน้าเว็บผลการทดลองที่มีตัวเลือกการแสดงผล 3 แบบ คือ

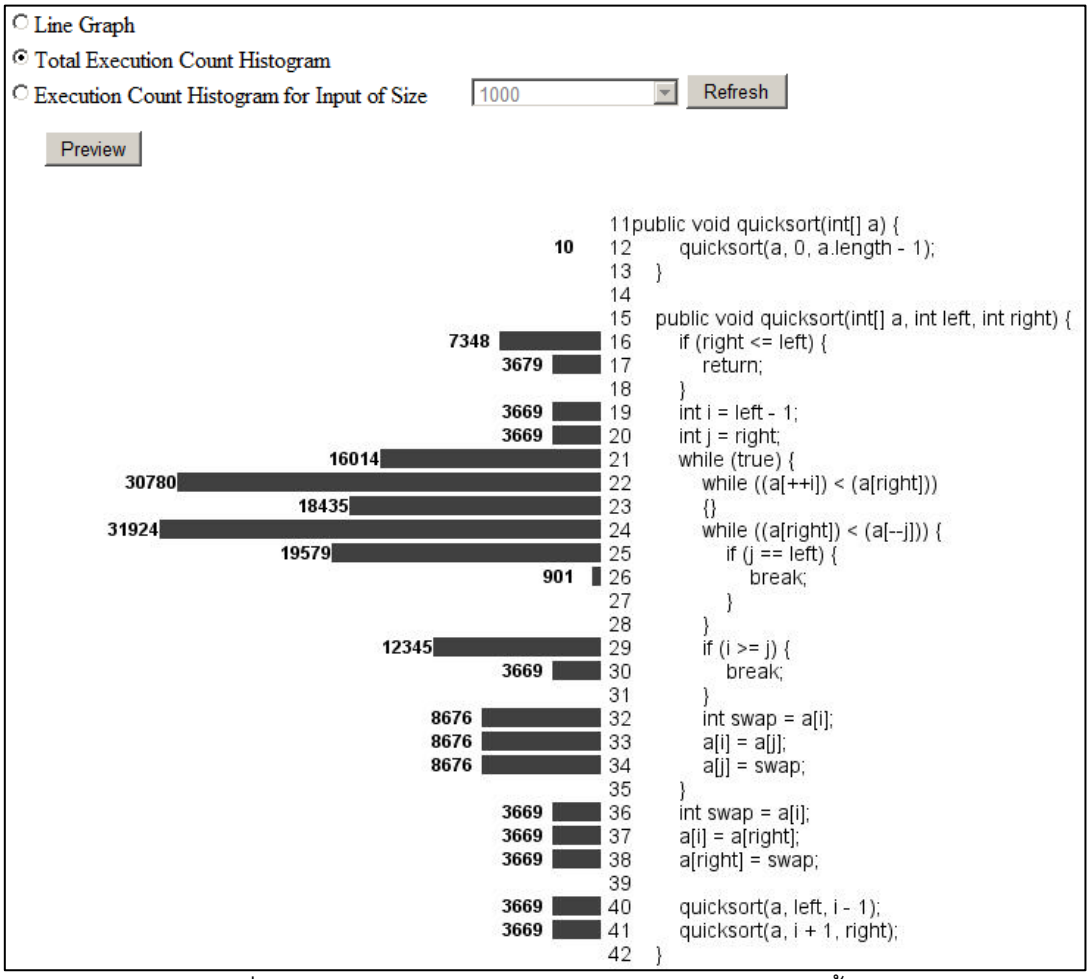
1. การแสดงผลลัพธ์รูปแบบกราฟเส้น (Line Graph)

2. การแสดงผลลัพธ์ในรูปแบบกราฟแท่งแนวขวางหรือฮิสโตแกรม แบบรวมผลลัพธ์จากการทดลองทั้งหมด (Total Execution Count Histogram)

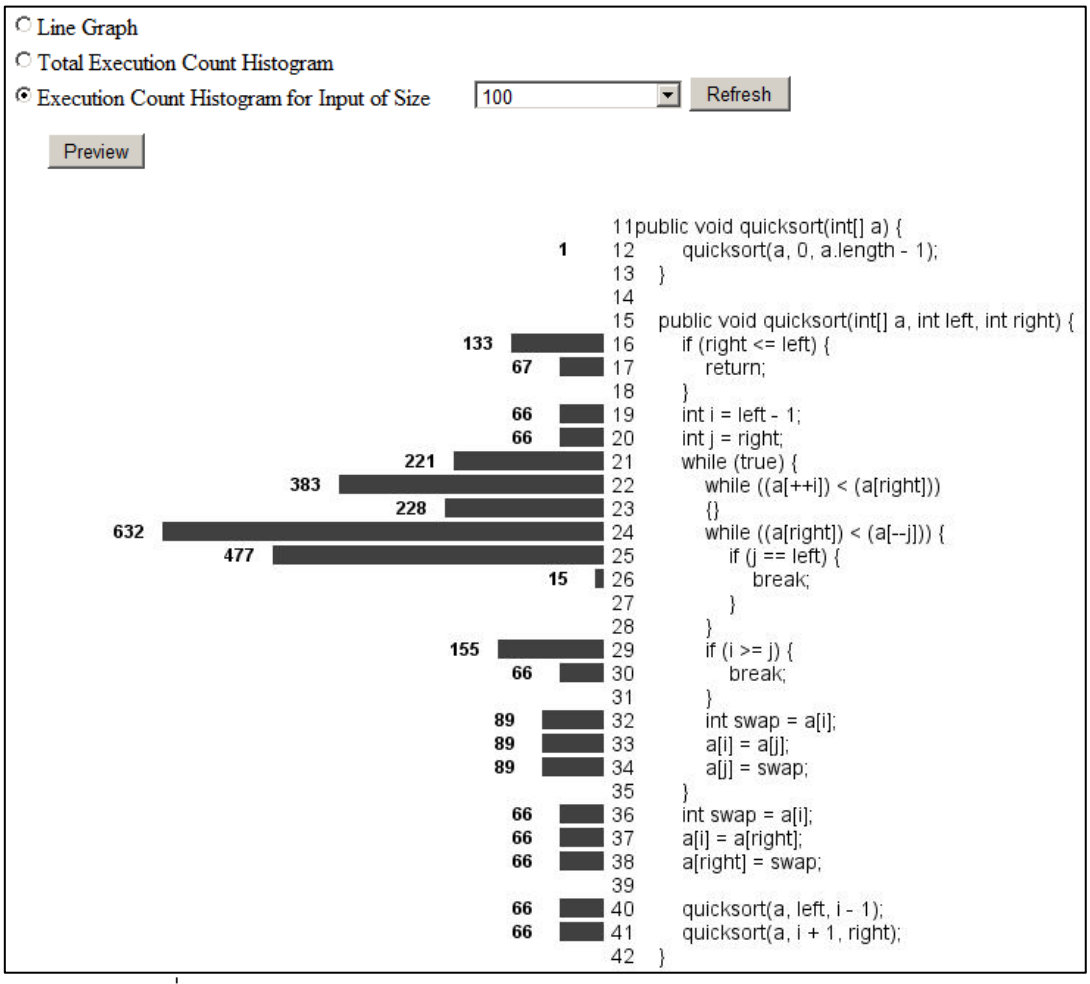
3. การแสดงผลลัพธ์ในรูปแบบกราฟแท่งแนวขวางหรือฮิสโตแกรม แบบแยกตามความกว้างของขนาดอาเรย์ที่ใช้ในการทดลอง (Execution Count Histogram for Input of Size)

โดยภาพที่ 3.11 ได้แสดงภาพตัวอย่างผลการทดลองในรูปแบบกราฟเส้น ภาพที่ 3.12 เป็นตัวอย่างผลการทดลองในรูปแบบกราฟฮิสโตแกรมโดยแสดงผลลัพธ์แบบรวมการทำงาน

ทั้งหมดของการทดลอง และภาพที่ 3.13 เป็นตัวอย่างผลการทดลองในรูปแบบกราฟฮิสโตแกรมที่แสดงผลลัพธ์ตามความกว้างของขนาดของอาเรย์ที่เลือก



ภาพที่ 3.12 ตัวอย่างผลลัพธ์รูปแบบฮิสโตแกรมแบบรวมทั้งหมด



ภาพที่ 3.13 ตัวอย่างผลลัพธ์รูปแบบฮิสโตแกรมแบบแยกตามขนาดของอาเรย์

3.2.2. การเรียกใช้งานผ่านบริการเว็บเซอร์วิส

ระบบเปิดให้บริการเว็บเซอร์วิสในรูปแบบ REST เซอร์วิส สามารถเรียกใช้งานผ่าน URL Address ของแต่ละ Web Browser ได้โดยตรง หรือสามารถเรียกใช้งานด้วยรหัสโปรแกรมในภาษาต่างๆ โดยกำหนด URL มีรูปแบบดังนี้

http://<servername>/AAS/AASServ.svc/profile?attr={JSON Value}

โดย URL ประกอบไปด้วย

- <servername> : ชื่อของเครื่องแม่ข่ายที่เว็บเซอร์วิสติดตั้งอยู่ หรือชื่อเว็บ
- {JSON Value} : บรรยายรูปแบบการทดลองด้วย JSON String [13]

ส่วนสำคัญคือส่วนของ {JSON Value} หรือส่วนที่อยู่หลังเครื่องหมาย '?' โดยผู้ใช้งานจะต้องระบุค่าตามหลังเครื่องหมาย '=' ดังตัวอย่างคือ {JSON Value} ให้ถูกต้อง ผู้ใช้งานจะต้องกำหนดรูปแบบการทดลองในรูปแบบ JSON String [13] ซึ่งประกอบด้วย

- codeMethod : เป็นข้อมูลประเภทสตริง (String) รหัสเมธ็อดที่ใช้ในการทดลอง
- experimentName : เป็นข้อมูลประเภทสตริง (String) ระบุชื่อที่ใช้ในการทดลอง
- strInputClsArray : เป็นข้อมูลประเภทสตริงอาเรย์ (String[]) เพื่อระบุคลาสของข้อมูลขาเข้า ผู้ใช้งานสามารถระบุคลาสข้อมูลขาเข้าได้มากกว่า 1 คลาส ได้แก่
  - jprofile.util.RandomIntArray.class
  - jprofile.util.SortedIntArray.class
  - jprofile.util.ReverseIntArray.class
  - jprofile.util.RandomDoubleArray.class
  - jprofile.util.SortedDoubleArray.class
  - jprofile.util.ReverseDoubleArray.class
- seedVal : เป็นจำนวนเต็มที่ระบุค่าเริ่มต้นที่ใช้ผลิตเลขสุ่มในการสร้างข้อมูลขาเข้าสำหรับใช้ในการทดลอง
- fromVal : เป็นจำนวนเต็มที่ระบุปริมาณข้อมูลเริ่มต้นที่ใช้ในการทดลอง
- toVal : เป็นจำนวนเต็มที่ระบุปริมาณข้อมูลมากที่สุดที่ใช้ในการทดลอง
- stepVal : เป็นจำนวนเต็มที่ระบุค่าที่เพิ่มขึ้นในแต่ละการทดลอง
- repeatVal : เป็นจำนวนเต็มที่ระบุจำนวนการทดลองซ้ำในแต่ละข้อมูลขาเข้าตามที่ระบุไว้
- axisXName : สตริง (String) ที่แทนข้อความกำกับแกน X
- axisYName : สตริง (String) ที่แทนข้อความกำกับแกน Y
- curveFitting : เป็นบูลีน (Boolean) ในการแสดงผลลัพธ์แบบปรับเส้นโค้ง
- emailAddress : ข้อมูลสตริง (String) เพื่อให้ส่งผลลัพธ์ไปยังอีเมลที่ระบุ

ตัวอย่าง JSON String ที่ใช้ในการเรียกใช้งานเว็บเซอร์วิส

```
{"codeMethod":
"public int max(int[] data) {
        \r\n int max = data[0];
        \r\n for (int i = 1; i \u003c data.length; i++) {
```

\r\n if (max \u003c data[i]) {
\r\n max = data[i];\r\n }
\r\n }
\r\n return max;
\r\n}",
"experimentName":"FindMax",
"strInputClsArray":["RandomIntArray"],
"seedVal":0,
"fromVal":100,
"toVal":1000,
"stepVal":50,
"repeatVal":200,
"axisXName":"X",
"axisYName":"Y",
"curveFitting":true,
"emailAddress":""}

สำหรับอักขระพิเศษที่ใช้ภายใน JSON String จะต้องอยู่ในรูป Unicode [14] เช่น

- เครื่องหมายน้อยกว่า (<) = “\u003c”
- เครื่องหมายมากกว่า (>) = “\u003e”
- เครื่องหมายลูกน้ำ (‘) = “\u0027”

เป็นต้น

โดยเมื่อนำ JSON String ไปใช้ตาม URL เว็บเซอร์วิสที่เตรียมไว้จะได้ URL ดังนี้

http://<servername>/AAS/AASServ.svc/profile?attr=

{"codeMethod":"public int max(int[] data) {\r\n int max = data[0];\r\n for (int i = 1; i \u003c data.length; i++) {\r\n if (max \u003c data[i]) {\r\n max = data[i];\r\n }\r\n } \r\n return max;\r\n}","experimentName":"FindMax","strInputClsArray":["RandomIntArray"],"seedVal":0,"fromVal":100,"toVal":1000,"stepVal":50,"repeatVal":200,"axisXName":"X","axisYName":"Y","curveFitting":true,"emailAddress":""}

เมื่อการทดลองเสร็จสิ้นระบบจะส่งผลลัพธ์กลับมาในรูปแบบ JSON String ประกอบด้วย

- urlResult : เป็นข้อมูลสตริง (String) ระบุ URL ของภาพผลลัพธ์ (ดังตัวอย่างภาพที่ 3.11, 3.12 และ 3.13)
- log : เป็นข้อมูลสตริง (String) แสดงข้อมูลการทำงาน เช่น ข้อความ Error หรือการทำงานของระบบ

ตัวอย่างผลลัพธ์ในรูปแบบ JSON String

{"urlResult":"http:\\localhost\\AlgorithmAnalysisService\\Output.aspx?expName=Exp20120327034520",
"log":"Error:java.io.FileNotFoundException: c:\\windows\\system32\\inetsrv\\src\\JavaExperiment\\Exp20120327034520.java (The system cannot find the path specified)\nError:The process cannot access the file \u0027C:\\inetpub\\wwwroot\\AlgorithmAnalysisService\\Output\\Log\\Log_Exp20120327034520.txt\u0027 because it is being used by another process.\n"}

บทที่ 4

ตัวอย่างการใช้งาน

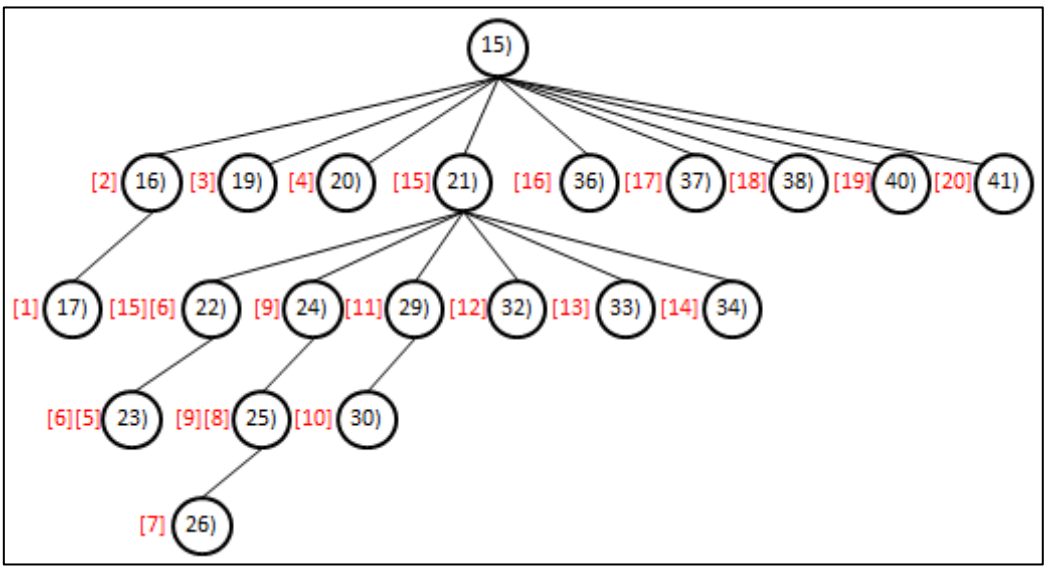
งานวิจัยนี้มีวัตถุประสงค์เพื่อออกแบบและพัฒนาเครื่องมือที่ช่วยในการวิเคราะห์อัลกอริทึมเชิงทดลอง เพื่อใช้ในการศึกษาและค้นคว้าอย่างมีประสิทธิภาพ โดยใช้วิธีการแทรกคำสั่งนับในรหัสต้นฉบับทุกคำสั่งอย่างอัตโนมัติ ดังนั้นในบทนี้จะกล่าวถึงตัวอย่างการทดลองและผลลัพธ์ที่ได้จากการทดลอง

4.1. การเรียงลำดับข้อมูลแบบเร็ว (Quicksort)

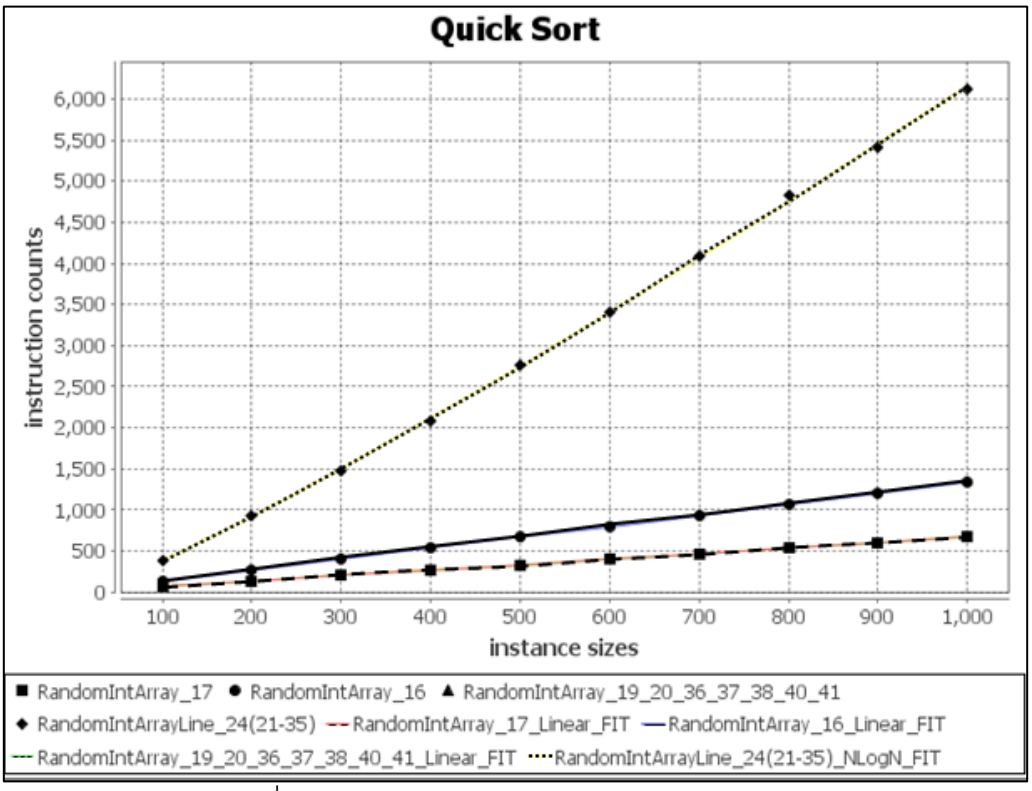
การเรียงลำดับข้อมูลแบบเร็ว (Quicksort) ทำงานโดยเลือกข้อมูลจากกลุ่มขึ้นมา 1 ค่า เพื่อใช้เป็นคีย์หลักในการแบ่งกลุ่มข้อมูล โดยเลือกข้อมูลจากตัวสุดท้ายของชุดข้อมูล หลังจากแบ่งกลุ่มข้อมูล ส่วนที่อยู่หน้าของค่าที่เลือกจะมีค่าน้อยกว่าหรือเท่ากับค่าที่เลือก ส่วนที่อยู่หลังจะมีค่ามากกว่า หลังจากนั้นนำแต่ละส่วนไปแบ่งส่วนย่อยด้วยวิธีเดียวกันจนไม่สามารถแบ่งย่อยได้อีก

```
12)     public void quicksort(int[] a) {
13)         quicksort(a, 0, a.length - 1);
14)     }
15)     public void quicksort(int[] a, int left, int right) {
16)         if (right <= left) {
17)             return;
18)         }
19)         int i = left - 1;
20)         int j = right;
21)         while (true) {
22)             while ((a[++i]) < (a[right]))
23)             {}
24)             while ((a[right]) < (a[--j])) {
25)                 if (j == left) {
26)                     break;
27)                 }
28)             }
29)             if (i >= j) {
30)                 break;
31)             }
32)             int swap = a[i];
33)             a[i] = a[j];
34)             a[j] = swap;
35)         }
36)         int swap = a[i];
37)         a[i] = a[right];
38)         a[right] = swap;
39)
40)         quicksort(a, left, i - 1);
41)         quicksort(a, i + 1, right);
42)     }
```

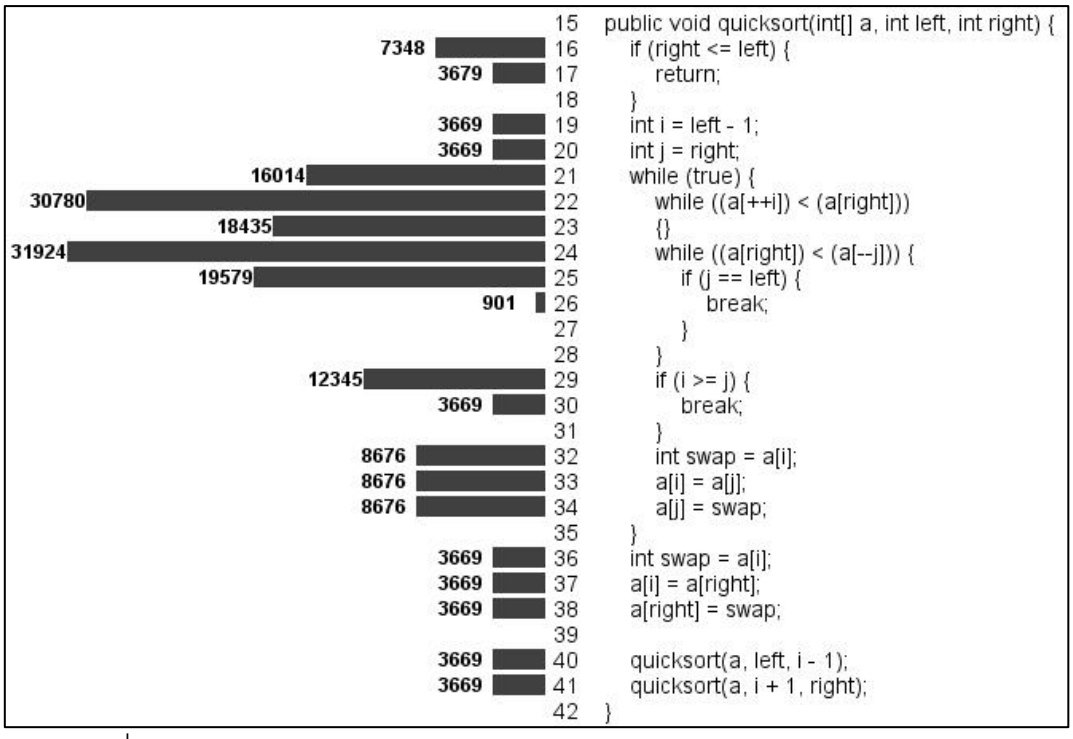
รหัสที่ 4.1 รหัสอัลกอริทึม Quicksort



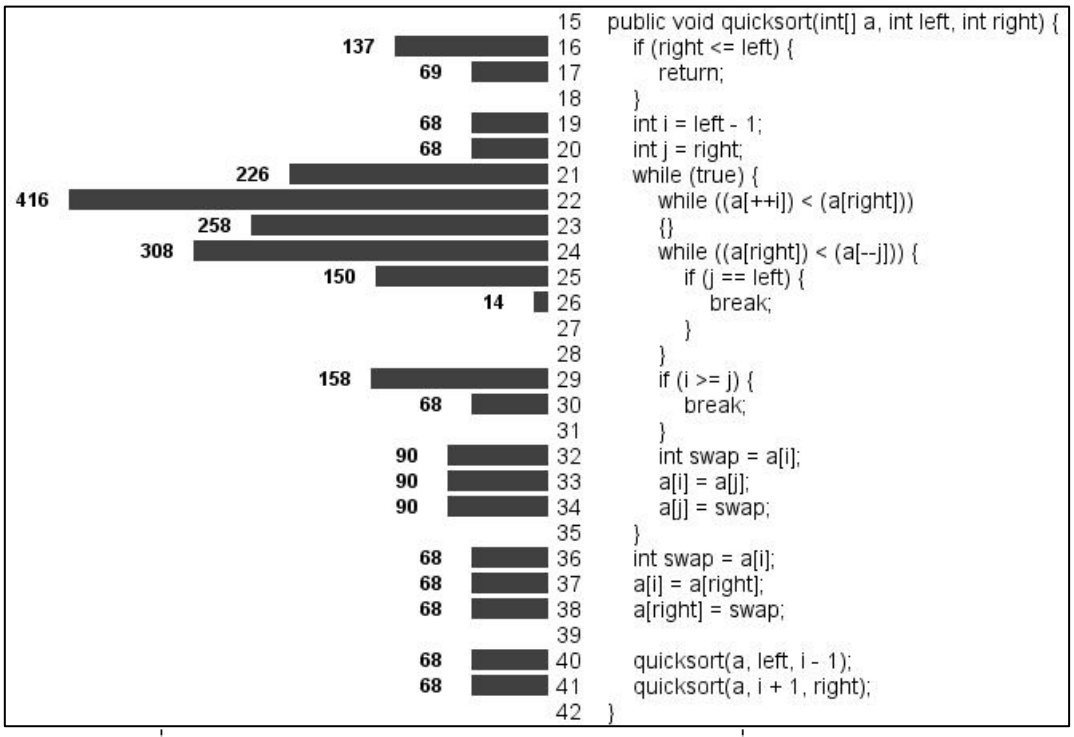
ภาพที่ 4.1 รูปแบบ AST ของรหัส Quicksort



ภาพที่ 4.2 ผลลัพธ์ในรูปแบบกราฟเส้นของ Quicksort



ภาพที่ 4.3 ผลลัพธ์ในรูปแบบกราฟฮิสโตแกรมแบบรวมผลการทำงานของ Quicksort



ภาพที่ 4.4 ผลลัพธ์ในรูปแบบกราฟฮิสโตแกรมช่วงอาเรย์ที่ 100 ของ Quicksort

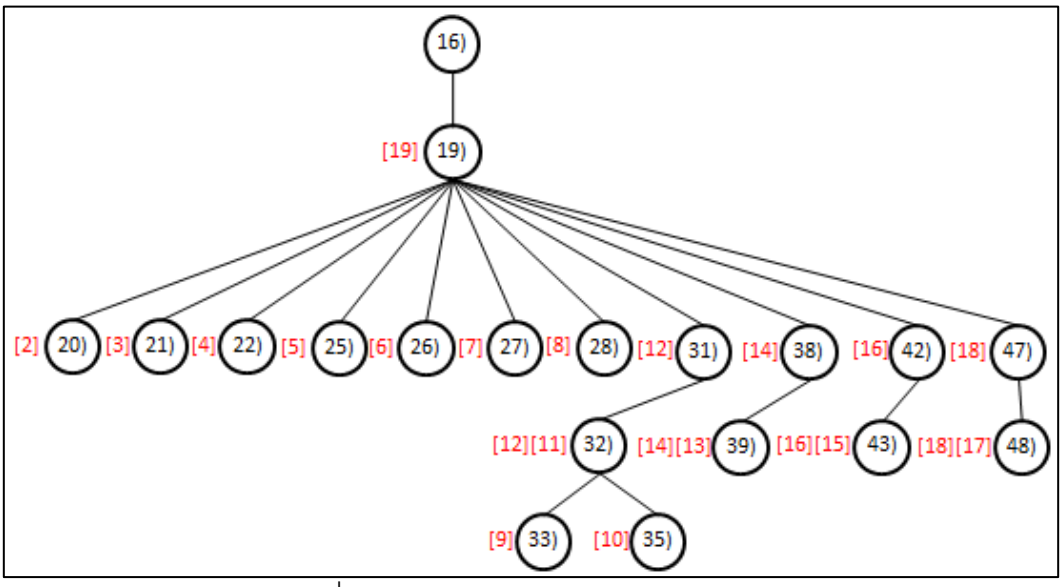
ภาพที่ 4.1 แสดง AST ของรหัสที่ 4.1 แต่ละโหนดแทนแต่ละบรรทัดของรหัสและแสดงการแทรกคำสั่งนับกำกับด้านหน้าของโหนด ผลลัพธ์ที่ได้จากภาพที่ 4.2 แสดงกราฟเส้นสะท้อนอัตราการเติบโตของการเรียงลำดับข้อมูลแบบเร็ว (Quicksort) จากรหัสที่ 4.1 โดยมีการเติบโตเป็นแบบ $O(n \log n)$ ผลลัพธ์การทำงานที่เด่นชัดที่สุดคือบรรทัดที่ 24 ซึ่งเป็นตัวแทนของรหัสลูปช่วงบรรทัดที่ 21-35 (ภาพที่ 4.2 ตำแหน่งบนกราฟเส้นคือ RandomIntArrayLine_24(21-25) และอัตราการเติบโตคือ RandomIntArrayLine_24(21-25)_NLogN_FIT) โดยภาพที่ 4.3 แสดงให้เห็นถึงความหนาแน่นของการทำงานของรหัสต้นฉบับของอัลกอริทึมแบบรวมผลลัพธ์การทำงาน โดยบรรทัดที่ 24 (31924) มีจำนวนครั้งการทำงานที่มากที่สุดที่ใช้เป็นตัวแทนในการแสดงผลกราฟในภาพที่ 4.2 ซึ่งการทำงานของบรรทัดที่ 24 ประกอบไปด้วยจำนวนครั้งการทำงานของบรรทัดที่ 25 (19579) และบรรทัดที่ 29 (12345) เป็นการเปรียบเทียบค่าในอาเรย์ ส่วนในภาพที่ 4.4 เป็นการแสดงผลลัพธ์ในช่วงของขนาดอาเรย์ที่ 100 จะเห็นว่าการทำงานบรรทัดที่ 22 มีการทำงานที่มากครั้งว่าบรรทัดที่ 24 โดยการทำงานของบรรทัดที่ 22 (416) ประกอบไปด้วยจำนวนครั้งการทำงานของบรรทัดที่ 23 (258) และ 29 (158) เป็นการเปรียบเทียบข้อมูลเริ่มจากทางซ้ายของอาเรย์ จะเห็นว่าทั้งภาพที่ 4.3 และ 4.4 แสดงให้เห็นจำนวนครั้งการเปรียบเทียบข้อมูลที่มากกว่าการสลับข้อมูล

4.2. การเรียงลำดับข้อมูลแบบประสาน (Merge Sort)

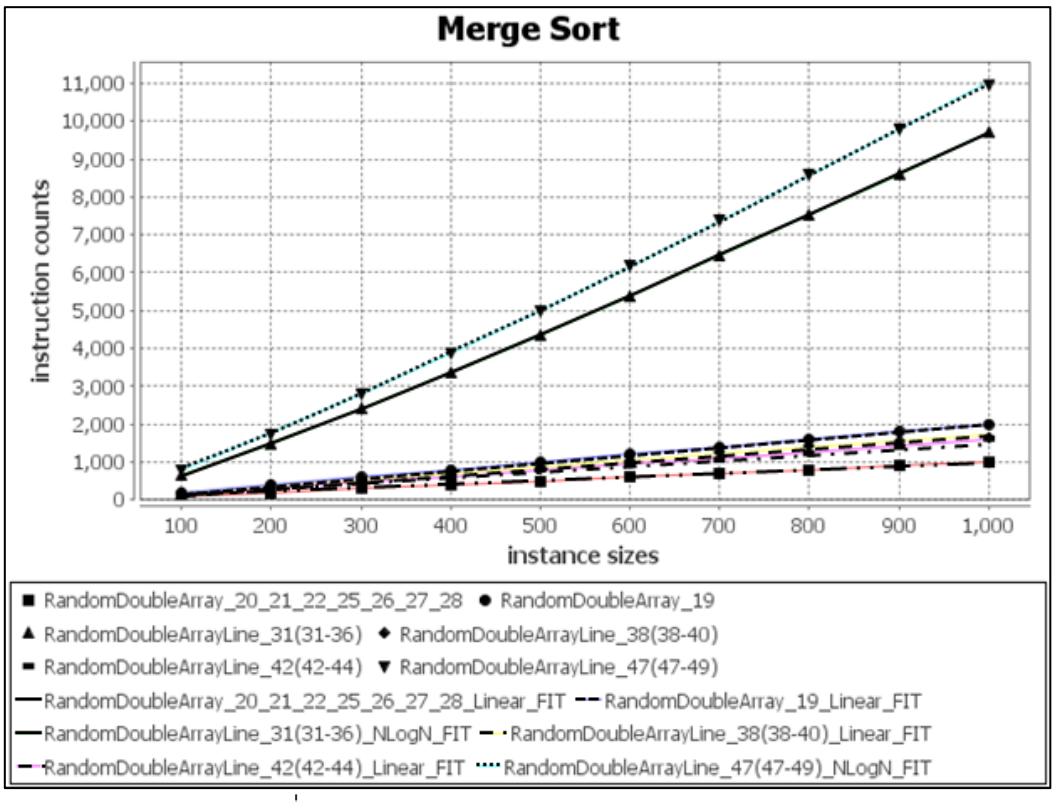
การจัดเรียงแบบประสาน เป็นการนำกลุ่มข้อมูลที่ถูกจัดเรียงลำดับแล้ว มาประสานให้เป็นชุดเดียวกัน โดยการประสานกันเป็นกลุ่มใหม่นั้นจะทำการจัดเรียงภายในของแต่ละกลุ่มด้วย โดยเริ่มต้นการประสานจากเป็นคู่ และนำแต่ละคู่มาประสานกันเป็นกลุ่มใหม่ ทำเช่นนี้จนเหลือข้อมูลเพียงกลุ่มเดียว

```
11) public static void mergeSort( double [ ] a ) {
12)     double [ ] tmpArray = new double[ a.length ];
13)     mergeSort( a, tmpArray, 0, a.length - 1 );
14) }
15)
16) private static void mergeSort( double [ ] a, double [ ] tmpArray,
17)         int left, int right ) {
18)
19)     if( left < right ) {
20)         int center = ( left + right ) / 2;
21)         mergeSort( a, tmpArray, left, center );
22)         mergeSort( a, tmpArray, center + 1, right );
23)
24)         //MERGE
25)         int rightPos = center + 1;
26)         int leftEnd = rightPos - 1;
27)         int tmpPos = left;
28)         int numElements = right - left + 1;
29)
30)         // Main loop
31)         while( left <= leftEnd && rightPos <= right ){
32)             if( a[ left ] <=  a[ rightPos ]  )
33)                 tmpArray[ tmpPos++ ] = a[ left++ ];
34)             else
35)                 tmpArray[ tmpPos++ ] = a[ rightPos++ ];
36)         }
37)
38)         while( left <= leftEnd ){    // Copy rest of first half
39)             tmpArray[ tmpPos++ ] = a[ left++ ];
40)         }
41)
42)         while( rightPos <= right ){  // Copy rest of right half
43)             tmpArray[ tmpPos++ ] = a[ rightPos++ ];
44)         }
45)
46)         // Copy tmpArray back
47)         for( int i = 0; i < numElements; i++, right-- ){
48)             a[ right ] = tmpArray[ right ];
49)         }
50)     }
51) }
```

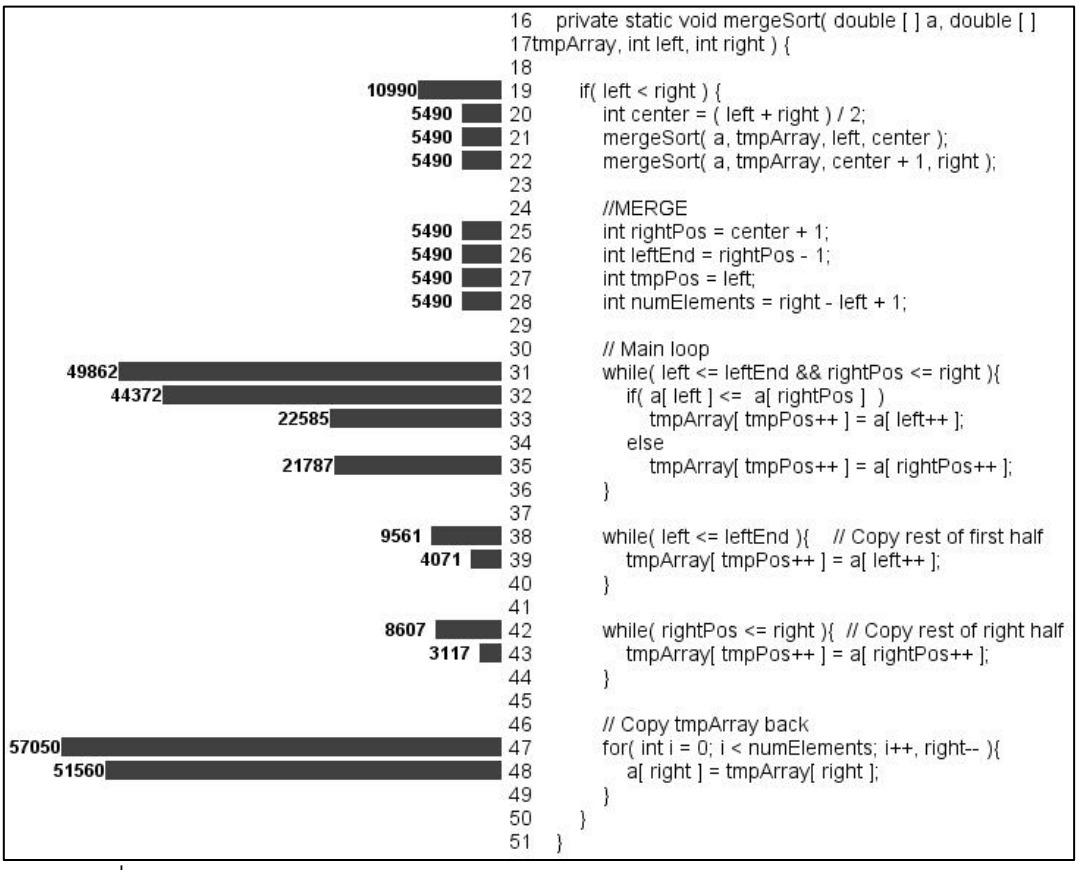
รหัสที่ 4.2 รหัสอัลกอริทึม Merge Sort



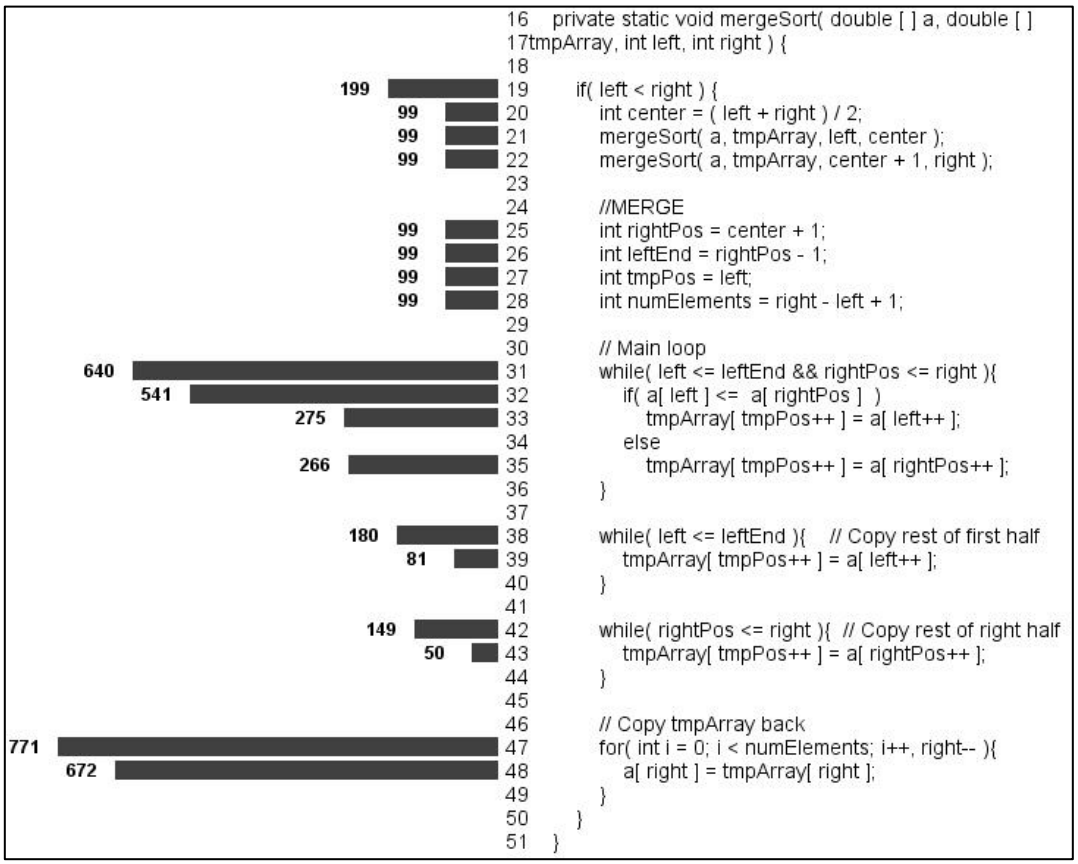
ภาพที่ 4.5 รูปแบบ AST ของรหัส Merge Sort



ภาพที่ 4.6 ผลลัพธ์รูปแบบกราฟเส้นของ Merge Sort



ภาพที่ 4.7 ผลลัพธ์ในรูปแบบกราฟฮิสโตแกรมแบบรวมผลการทำงานของ Merge Sort



ภาพที่ 4.8 ผลลัพธ์ในรูปแบบกราฟฮิสโตแกรมช่วงอาเรย์ที่ 100 ของ Merge Sort

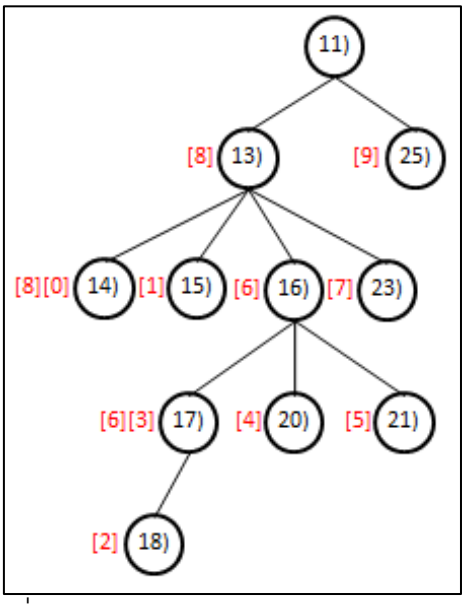
ภาพที่ 4.5 แสดง AST ของรหัสที่ 4.2 แต่ละโหนดแทนแต่ละบรรทัดของรหัสและแสดงการแทรกคำสั่งนับกำกับด้านหน้าของโหนด ผลลัพธ์ภาพที่ 4.6 แสดงให้เห็นว่ารหัสต้นฉบับของการเรียงลำดับข้อมูลแบบประสาน (Merge Sort) มีการเติบโตของเส้นกราฟ RandomIntArrayLine_47(47-49)_NLogN_FIT เป็น $O(n \log n)$ โดยมีบรรทัดที่ 47 เป็นตัวแทนที่สะท้อนอัตราการเติบโตของรหัสต้นฉบับ ซึ่งเป็นบรรทัดตัวแทนของรหัสลูปตั้งแต่บรรทัดที่ 47-49 จากภาพที่ 4.8 จะเห็นได้ว่าบรรทัดที่ 47 มีการทำงานจำนวนครั้งมากที่สุด ภายในลูปประกอบไปด้วยจำนวนครั้งการทำงานของบรรทัดที่ 48 (672) และบรรทัดที่ 25 (99) ที่เป็นบรรทัดการกำหนดค่าตัวแปรเมื่อเข้าเงื่อนไข if ที่บรรทัด 19 ซึ่งบรรทัดที่ 48 เป็นการสลับข้อมูล

4.3. **การเรียงลำดับข้อมูลแบบแทรก** (Insertion Sort)

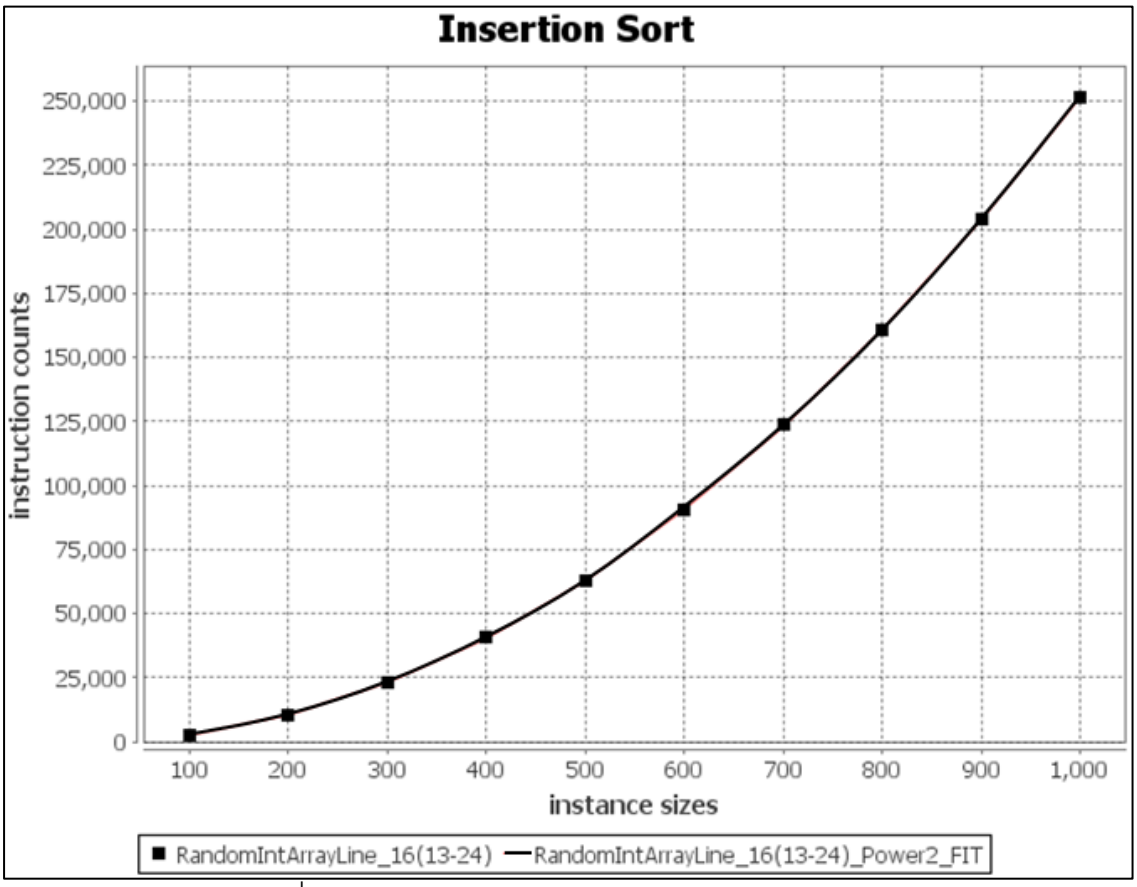
การเรียงลำดับข้อมูลแบบแทรก (Insertion Sort) เป็นการเรียงลำดับโดยพิจารณาค่าตั้งแต่ลำดับที่ 2 จนถึง N ในการวนทำงานในแต่ละครั้ง จะหาตำแหน่งที่เหมาะสม เพื่อแทรกค่าที่พิจารณาลงในตำแหน่งที่เหมาะสมสำหรับชุดข้อมูลนั้นๆ

```
11)     public int[] insertionSort(int[] data) {
12)
13)         for (int i = 1; i < data.length; i++) {
14)             int j = i;
15)             int temp = data[i];
16)             while (j > 0) {
17)                 if (data[j - 1] < temp) {
18)                     break;
19)                 }
20)                 data[j] = data[j - 1];
21)                 j--;
22)             }
23)             data[j] = temp;
24)         }
25)         return data;
26)     }
```

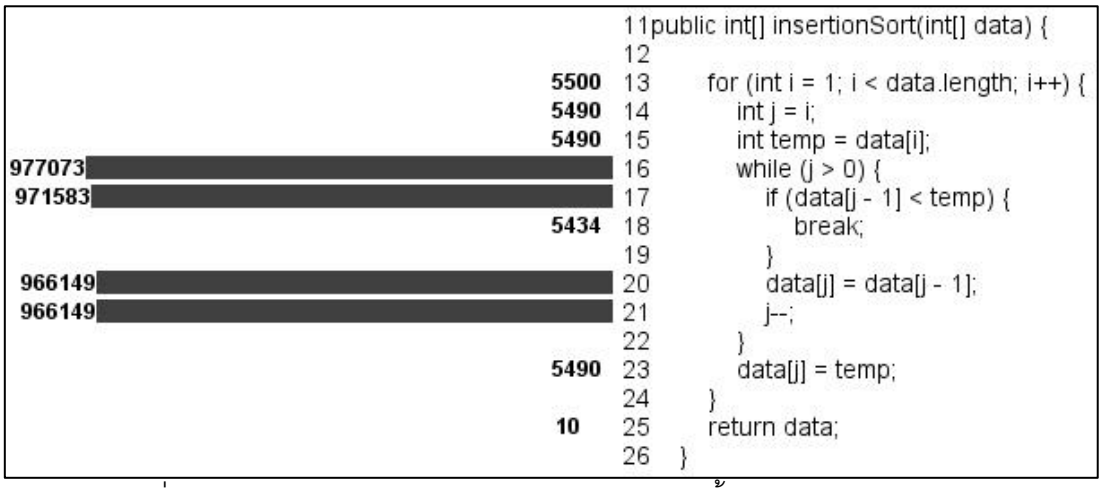
รหัสที่ 4.3 รหัสอัลกอริทึม Insertion Sort



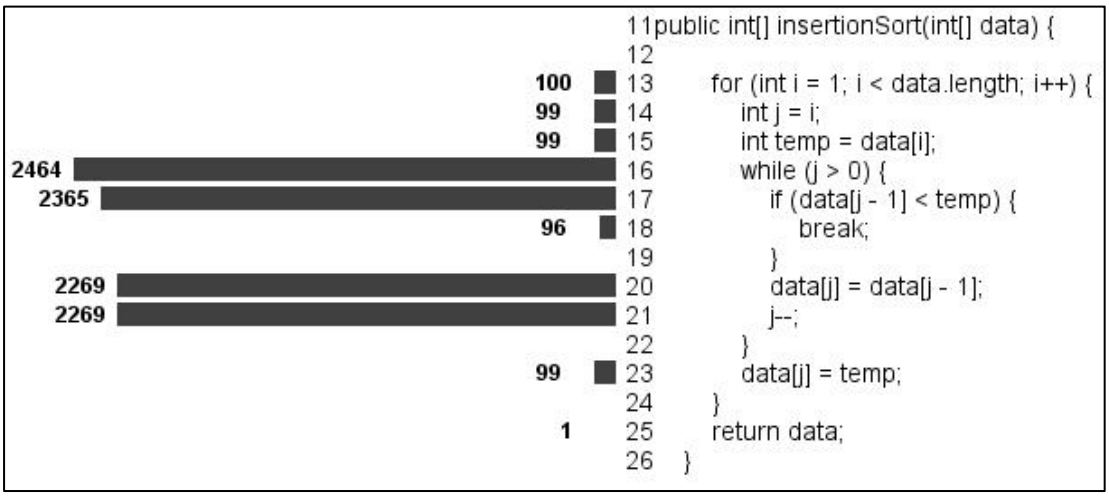
ภาพที่ 4.9 รูปแบบ AST ของรหัส Insertion Sort



ภาพที่ 4.10 ผลลัพธ์รูปแบบกราฟเส้นของ Insertion Sort



ภาพที่ 4.11 ผลลัพธ์รูปแบบกราฟฮิสโตแกรมแบบรวมทั้งหมดของ Insertion Sort



ภาพที่ 4.12 ผลลัพธ์รูปแบบกราฟฮิสโตแกรมช่วงความกว้างอาเรย์ที่ 100 ของ Insertion Sort

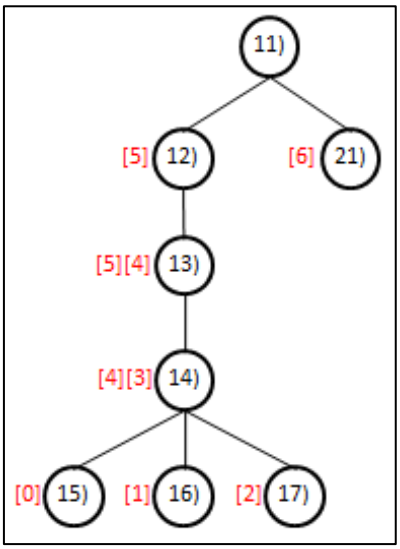
ภาพที่ 4.9 แสดง AST ของรหัสที่ 4.3 แต่ละโหนดแทนแต่ละบรรทัดของรหัสและแสดงการแทรกคำสั่งนับกำกับด้านหน้าของโหนด ผลลัพธ์ภาพที่ 4.10 เป็นผลลัพธ์จากการทดลองของรหัสที่ 4.3 มีตัวแทนของรหัสต้นฉบับ คือ RandomIntArrayLine_16(13-24) และมีอัตราการเติบโตแบบ RandomIntArrayLine_16(13-24)_Power2_FIT หมายถึงการทดลองนี้มีอัตราการเติบโตเป็น O (n^2) โดยในภาพที่ 4.11 และ 4.12 เป็นผลลัพธ์ในรูปแบบกราฟฮิสโตแกรมที่ทำให้เห็นการทำงานที่ชัดเจนมากขึ้น ในภาพที่ 4.12 ภายใต้ลูป for ของบรรทัดที่ 13 พบว่าลูป while บรรทัดที่ 16 (2464) มีการทำงานจำนวนครั้งมากที่สุด และภายใต้ลูป while นั้นจะเห็นได้ว่าการทำงานในส่วนของการเปรียบเทียบข้อมูล (บรรทัดที่ 17 (2365)) และการสลับข้อมูล (บรรทัดที่ 20 (2269)) จะมีปริมาณการทำงานที่ใกล้เคียงกัน

4.4. **การเรียงลำดับข้อมูลแบบบับเบิล** (Bubble Sort)

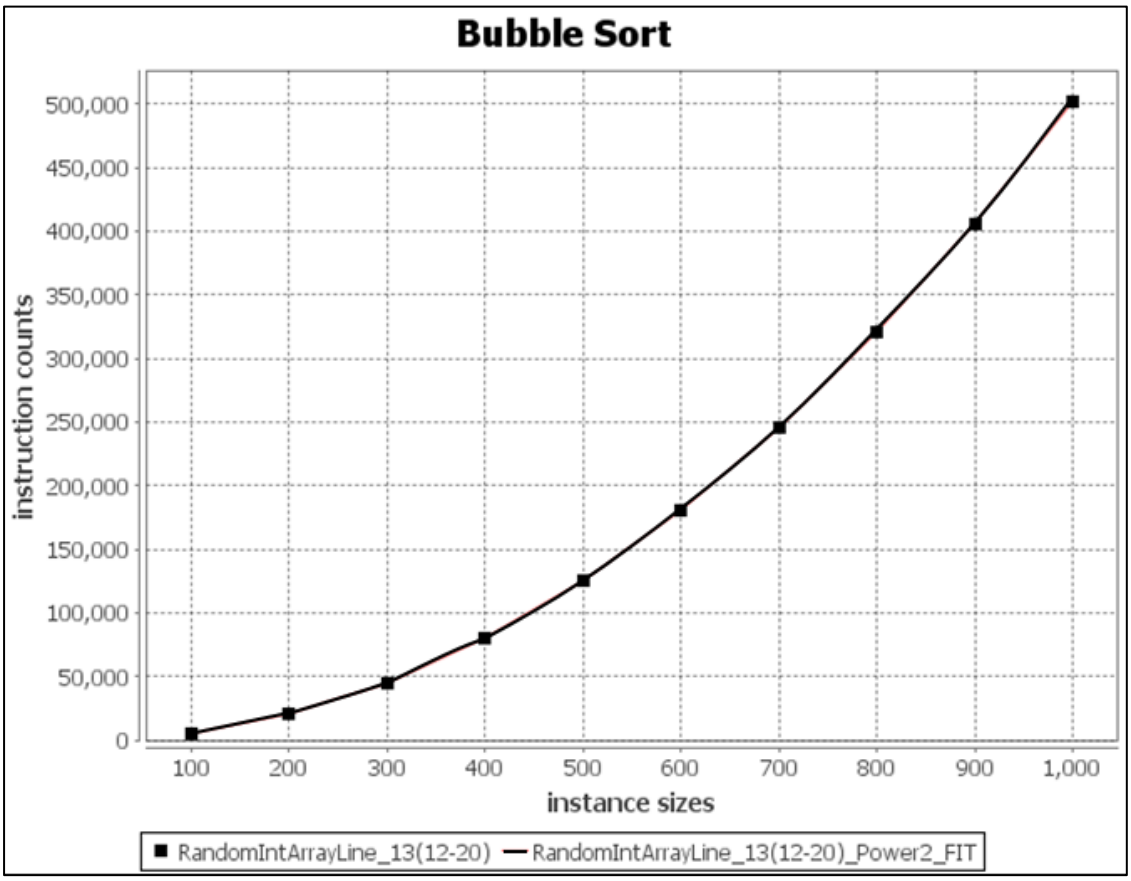
การเรียงลำดับข้อมูลแบบบับเบิล (Bubble Sort) เป็นการจัดเรียงโดยเปรียบเทียบข้อมูล 2 ลำดับที่ติดกัน ถ้าลำดับของข้อมูลไม่ถูกต้องจึงสลับตำแหน่ง เปรียบเทียบลักษณะนี้ไปจนกระทั่งการเปรียบเทียบข้อมูลทั้งหมดจะไม่มีการสลับลำดับข้อมูลอีก

```
11)     public int[] bubbleSort(int[] data) {
12)         for (int i = 0; i < data.length; i++) {
13)             for (int j = i; j < data.length; j++) {
14)                 if (data[i] > data[j]) {
15)                     int temp = data[i];
16)                     data[i] = data[j];
17)                     data[j] = temp;
18)                 }
19)             }
20)         }
21)         return data;
22)     }
```

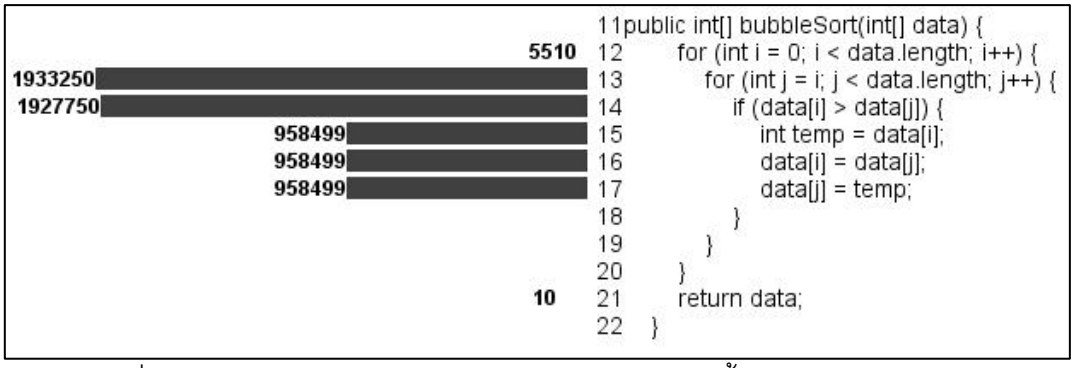
รหัสที่ 4.4 รหัสอัลกอริทึม Bubble Sort



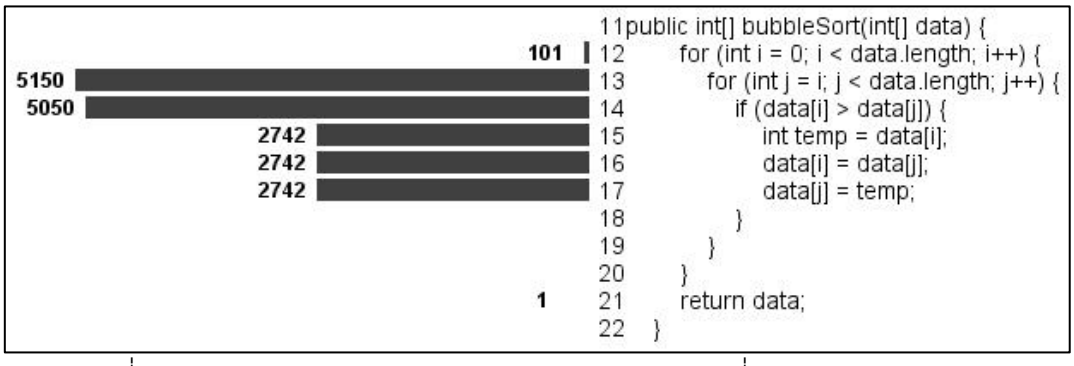
ภาพที่ 4.13 รูปแบบ AST ของรหัส Bubble Sort



ภาพที่ 4.14 ผลลัพธ์รูปแบบกราฟเส้นของ Bubble Sort



ภาพที่ 4.15 ผลลัพธ์รูปแบบกราฟฮิสโตแกรมแบบผลรวมทั้งหมดของ Bubble Sort



ภาพที่ 4.16 ผลลัพธ์รูปแบบกราฟฮิสโตแกรมของช่วงอาเรย์ที่ 100 ของ Bubble Sort

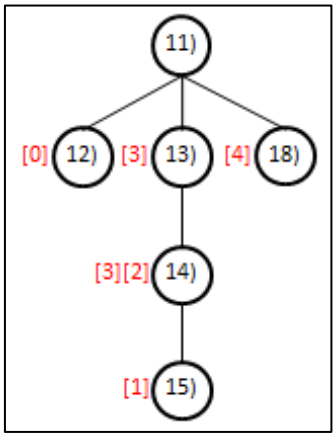
ภาพที่ 4.13 แสดง AST ของรหัสที่ 4.3 แต่ละโหนดแทนแต่ละบรรทัดของรหัสและแสดงการแทรกคำสั่งนับกำกับด้านหน้าของโหนด ผลลัพธ์ภาพที่ 4.14 แสดงผลลัพธ์รูปแบบกราฟเส้นของ Bubble Sort โดยมีตัวแทนรหัสคือ RandomIntArrayLine_13(12-20) และมีอัตราการเติบโต RandomIntArrayLine_13(12-20)_Power2_FIT หรืออัตราการเติบโตเป็น O (n^2) ภาพที่ 4.16 แสดงให้เห็นการทำงานที่มากที่สุดคือบรรทัดที่ 13 (5150) เป็นบรรทัดการทำงานของลูป for ภายใต้การทำงานของลูปประกอบไปด้วยการเปรียบเทียบข้อมูล และการสลับข้อมูลภายในอาเรย์ โดยการเปรียบเทียบมีปริมาณการทำงานที่มากที่สุด คือบรรทัดที่ 14 (5050)

4.5. การหาค่าสูงที่สุด (Find Max)

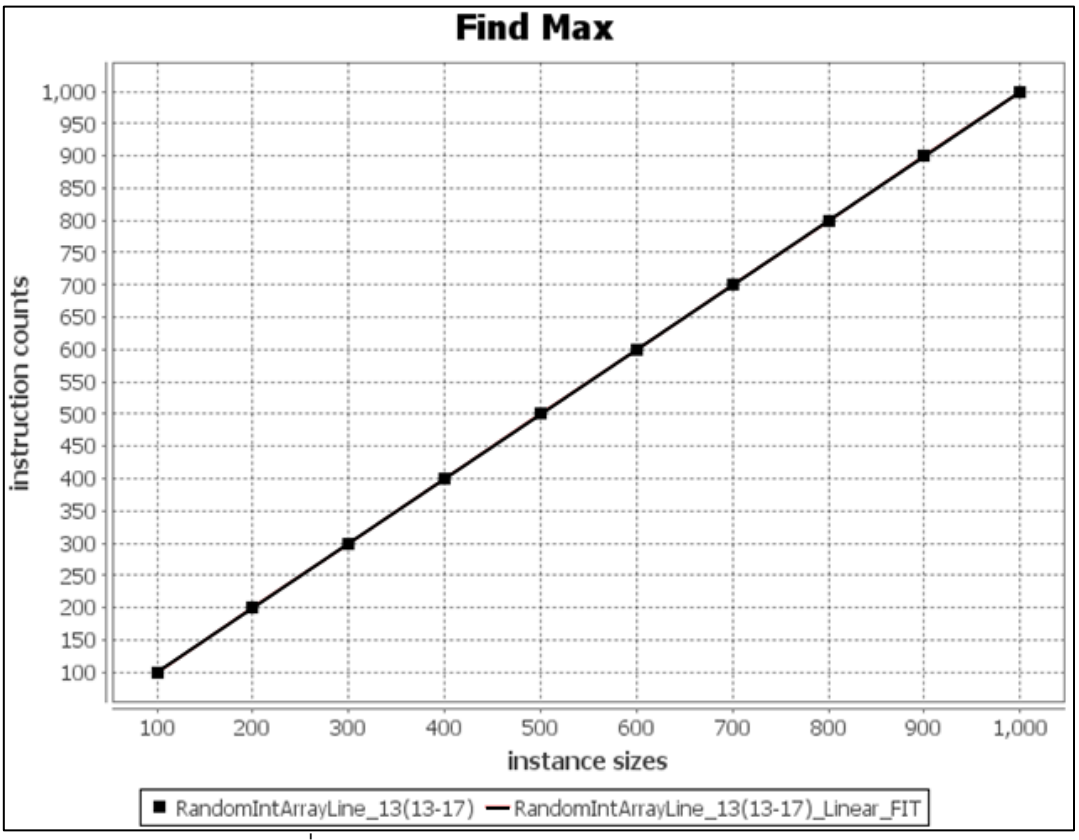
การหาค่าสูงที่สุด (Find Max) เป็นการวนเปรียบเทียบค่าในชุดข้อมูลเพื่อหาค่าที่สูงที่สุด

```
11)     public int max(int[] data) {
12)         int max = data[0];
13)         for (int i = 1; i < data.length; i++) {
14)             if (max < data[i]) {
15)                 max = data[i];
16)             }
17)         }
18)         return max;
19)     }
```

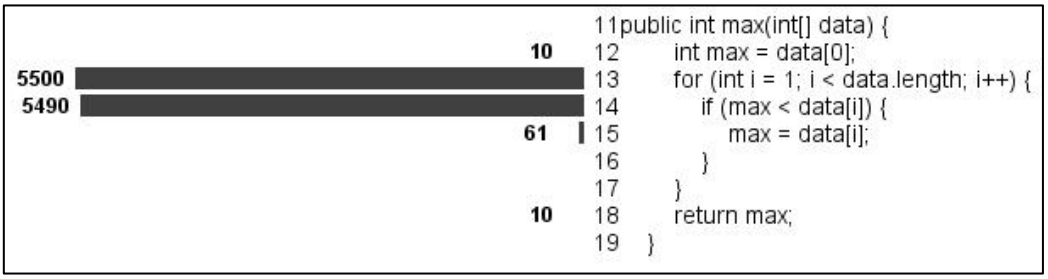
รหัสที่ 4.5 รหัสอัลกอริทึม Find Max



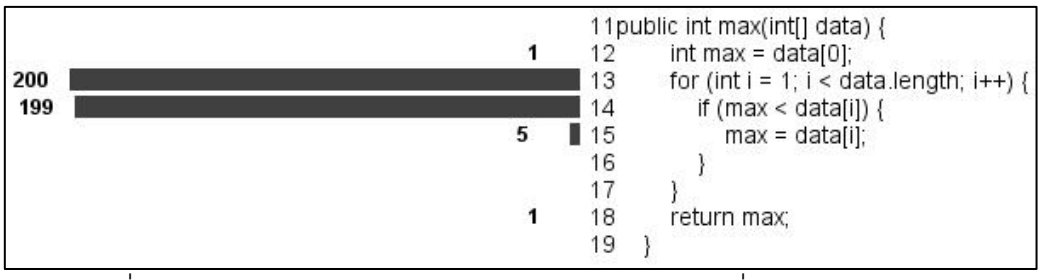
ภาพที่ 4.17 รูปแบบ AST ของรหัส Find Max



ภาพที่ 4.18 ผลลัพธ์รูปแบบกราฟเส้นของ Find Max



ภาพที่ 4.19 ผลลัพธ์รูปแบบกราฟฮิสโตแกรมแบบรวมผลของ Find Max



ภาพที่ 4.20 ผลลัพธ์รูปแบบกราฟฮิสโตแกรมของช่วงอาเรย์ที่ 200 ของ Find Max

ภาพที่ 4.17 แสดง AST ของรหัสที่ 4.3 แต่ละโหนดแทนแต่ละบรรทัดของรหัสและแสดงการแทรกคำสั่งนับกำกับด้านหน้าของโหนด ผลลัพธ์ภาพ 4.18 แสดงผลลัพธ์ด้วยตัวแทน RandomIntArrayLine_13(13-17) โดยการแสดงผลอัตราการเติบโตแบบ RandomIntArrayLine_13(13-17)_Linear_FIT หรืออัตราการเติบโตแบบ O (n) จากภาพที่ 4.19 แสดงให้เห็นว่าบรรทัดที่ 13 (5500) มีการจำนวนการทำงานมากที่สุด โดยการเปรียบเทียบค่ามีจำนวนการทำงานที่มากกว่าการกำหนดค่าสูงสุด

บทที่ 5

สรุปผลการวิจัย อภิปรายผล และข้อเสนอแนะ

5.1. สรุปผลการวิจัย

งานวิจัยนี้ได้พัฒนาเครื่องมือเพื่อช่วยในการวิเคราะห์อัลกอริทึมเชิงทดลอง เพื่อใช้ในการศึกษาและค้นคว้าประสิทธิภาพเชิงเวลาการทำงานของแต่ละอัลกอริทึม โดยที่ผู้วิเคราะห์ไม่จำเป็นต้องระบุบรรทัดตัวแทน เพื่อใช้เป็นตัวแทนในการสะท้อนการทำงานของแต่ละอัลกอริทึม แต่เครื่องมือนี้จะช่วยให้ผู้วิเคราะห์สามารถทราบบรรทัดตัวแทน และการทำงานของรหัสต้นฉบับผ่านผลลัพธ์ที่ได้จากการทดลอง ทำให้ผู้วิเคราะห์ทราบถึงปัญหาของแต่ละรหัสต้นฉบับและสามารถพัฒนารูปแบบรหัสต้นฉบับของแต่ละอัลกอริทึมได้อย่างมีประสิทธิภาพมากขึ้น

รหัสอัลกอริทึม	ปริมาณข้อมูลขาเข้า				จำนวนการทดลอง	เวลาการทดลอง	
	ขนาดเริ่มต้น	ขนาดสิ้นสุด	เพิ่มครั้งละ	วนซ้ำ		แบบระบุบรรทัดตัวแทน (วินาที)	แบบแทรกทุกบรรทัด (วินาที)
Quicksort	500	10000	500	100	100	3.94	5.93
Merge Sort	500	10000	500	100	100	6.8	10.47
Insertion Sort	500	10000	500	100	100	78.62	147.82
Bubble Sort	500	10000	500	100	100	220.42	351.31
Find Max	500	10000	500	100	100	3.51	3.78

ตารางที่ 5.1 เปรียบเทียบเวลาการทำงานระหว่างการทดลองด้วยวิธีการระบุบรรทัดตัวแทน และวิธีแทรกคำสั่งนับทุกบรรทัด

จากตารางที่ 5.1 แสดงให้เห็นถึงเวลาการทำงานของการทดลองระหว่างการระบุบรรทัดตัวแทนโดยไม่ผ่านเครื่องมืองานวิจัย และการทดลองโดยการโดยแทรกคำสั่งนับทุกบรรทัดของรหัสต้นฉบับผ่านเครื่องมืองานวิจัย พบว่าทั้งสองวิธีใช้เวลาการทดลองแตกต่างกัน โดยวิธีการแทรกคำสั่งนับทุกบรรทัดของรหัสต้นฉบับนั้นใช้เวลาการทดลองมากกว่าวิธีการระบุบรรทัด แต่งานวิจัยเน้นผลลัพธ์ในเรื่องของความถูกต้องในการวิเคราะห์ผลลัพธ์ที่ดีขึ้น เพื่อใช้ในการศึกษาในวิชาโครงสร้างข้อมูลและการศึกษาอัลกอริทึม และความง่ายต่อการเข้าถึงการทดลอง

งานวิจัยได้พัฒนารูปแบบการใช้งานให้ง่ายยิ่งขึ้น ด้วยรูปแบบการติดต่อและแสดงผลผ่านหน้าเว็บ ผู้วิเคราะห์จำเป็นต้องรู้ถึงส่วนต่างๆที่จำเป็นในการทดลอง และระบุข้อมูลการทดลองให้ถูกต้อง

5.2. **อภิปรายผล**

การวิเคราะห์อัลกอริทึมเชิงทดลองช่วยสะท้อนการทำงานที่เกิดขึ้นจริงในแต่ละอัลกอริทึม งานวิจัยนี้ได้แสดงวิธีการแทรกคำสั่งนับเพื่อเป็นตัวแทนในการทำงานของรหัสต้นฉบับในแต่ละบรรทัดอย่างอัตโนมัติ ทำการทดลองและแสดงผลลัพธ์ที่ทำให้ผู้วิเคราะห์ได้เห็นภาพของการทำงานชัดเจนมากขึ้น เพื่อนำผลลัพธ์ไปใช้ในการวิเคราะห์ออกแบบ และปรับปรุงวิธีการทำงานของรหัสอัลกอริทึม ก่อให้เกิดแนวคิดและวิธีการในการปรับปรุงอัลกอริทึมที่หลากหลาย รวมทั้งสามารถนำไปใช้ประกอบการเรียนการสอนในรายวิชาที่เกี่ยวข้อง

5.3. **ข้อเสนอแนะ**

5.3.1. ทำให้ใช้ได้กับภาษาโปรแกรมอื่น เช่น C++, C#.NET เป็นต้น

5.3.2. นับจำนวนครั้งรวมของคำสั่งต่าง ๆ ที่ทำในการเรียกแต่ละเมท็อดย่อย

5.3.3. นำเสนอผลการทำงานเพื่อช่วยการวิเคราะห์ในกรณีที่เขียนโปรแกรมแบบ recursive

5.3.4. นอกจากที่ระบบจะหาคำสั่งที่ทำมากสุดแล้ว ควรอนุญาตให้ผู้ใช้เลือกบางบรรทัดคำสั่งที่สนใจ หรือผลรวมของจำนวนครั้งของบรรทัดที่สนใจ

5.3.5. ให้ผู้ใช้ใส่สูตรการคำนวณเพื่อประมวลผลหลังการนับ เช่น นำจำนวนครั้งที่แต่ละคำสังทำงานมาผ่านฟังก์ชัน หารด้วยปริมาณข้อมูล เป็นต้น

รายการอ้างอิง

[1] R. Sedgewick and P. Flajolet. An Introduction to the Analysis of Algorithms. USA : Addison-Wesley Longman Publishing Co., 1996.

[2] D. Johnson. A Theoretician's Guide to the Experimental Analysis of Algorithms. AT&T Labs November 2001 : 1-36.

[3] Sun MicroSystems Inc. Anotation [Online]. 2011. Available from : http://java.sun.com/javase/6/docs/technotes/guides/language/annotations.html [2012,May 6]

[4] W. Nilla-or and S. Prasitjutrakul. JProfile101 : Controller for Experimental Analysis of Algorithms. NCIT2009, National Conference on Information Technology 2009.

[5] S. Liang and D. Viswanathan. Comprehensive Profiling Support in the Java Virtual Machine. The USENIX Association 1999.

[6] Sun MicroSystems Inc. Java Virtual Machine Profiler Interface (JVMPI) [Online]. 2011. Available from : http://download.oracle.com/javase/1.5.0/docs/guide/jvmpi/jvmpi.html [2012,May 6]

[7] Sun MicroSystems Inc. JVM Tool Interface [Online]. 2011. Available from : http://download.oracle.com/javase/1.5.0/docs/guide/jvmti/jvmti.html [2012,May 6]

[8] Sun Microsystems Inc. Abstract Syntax Tree [Online]. 2011. Available from : http://www.eclipse.org/articles/article.php?file=Article-JavaCodeManipulation_AST/index.html [2012,May 6]

[9] A. Duffy and T. Dowling. Algorithm Analysis. Department of Computer Science Technical Report Series National University of Ireland, 2003.

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

[11] M. Kuperberg, M. Krogmann and R. Reussner. ByCounter : Portable Runtime Counting of Bytecode Instructions and Method Invocations. Electronic Notes in Theoretical Computer Science 2008 : 1-15.

[12] Eclipse Foundation Inc. Eclipse IDE [Online]. 2001. Available from :

http://www.eclipse.org/ [2012,May 6]

[13] JSON (JavaScript Object Notation) [Online]. 2002. Available from : http://www.json.org/ [2012,May 6]

[14] Standard ECMA-262 [Online]. 1999. Available from : http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf [2012,May 6]

[15] I. Sanders. Empirical analysis of algorithms is easy (or is it?). School of Computer Science University of the Witwatersrand 2001 : 1-10.

[16] C. C. McGeoch. Experimental Analysis of Algorithms. NOTICES OF THE AMS 2001 : 48(3).

[17] I. Sanders. Teaching empirical analysis of algorithms. ACM SIGCSE Bulletin 2002 : 321-325.

[18] A. Duffy and T. Dowling. A Java API for Experimental Analysis of Algorithms. Computer Science Department NUI Maynooth Co Kildare, 2004.

ประวัติผู้เขียนวิทยานิพนธ์

นายธนินทร์ กระจังทอง เกิดเมื่อวันที่ 31 ธันวาคม พ.ศ. 2528 ที่จังหวัดกรุงเทพมหานคร เป็นบุตรชายคนแรก ของนายชาติชาย กระจังทอง และนางงามเพ็ญ กระจังทอง สำเร็จการศึกษาระดับปริญญาวิทยาศาสตรบัณฑิต สาขาวิชาวิทยาศาสตร์คอมพิวเตอร์ จากมหาวิทยาลัยราชภัฏสวนดุสิต ในปี พ.ศ. 2550 และได้เข้าศึกษาต่อในระดับปริญญาวิทยาศาสตรมหาบัณฑิต สาขาวิชาวิทยาศาสตร์คอมพิวเตอร์ คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย ในภาคการศึกษาต้น ปีการศึกษา 2552

ภาคผนวก

ภาคผนวก ก แทรกคำสั่งนับด้วย AST

ก.1 รหัสคำสั่งแปลงข้อมูลรหัสต้นฉบับเป็น AST และแทรกคำสั่งนับลงในแต่ละโหนดของ AST ต้นฉบับ และแปลง AST ผลลัพธ์กลับเป็นรหัสที่มีคำสั่งนับพร้อมสำหรับทดลองผ่าน JProfile101

```
package CodeCounter;

import java.io.BufferedOutputStream;
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.OutputStreamWriter;
import java.lang.reflect.InvocationTargetException;
import java.lang.reflect.Method;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.ListIterator;
import java.util.Map;

import org.eclipse.jdt.core.ICompilationUnit;
import org.eclipse.jdt.core.dom.AST;
import org.eclipse.jdt.core.dom.ASTNode;
import org.eclipse.jdt.core.dom.ASTParser;
import org.eclipse.jdt.core.dom.AbstractTypeDeclaration;
import org.eclipse.jdt.core.dom.Annotation;
import org.eclipse.jdt.core.dom.ArrayAccess;
import org.eclipse.jdt.core.dom.ArrayCreation;
import org.eclipse.jdt.core.dom.ArrayInitializer;
import org.eclipse.jdt.core.dom.Assignment;
import org.eclipse.jdt.core.dom.Block;
import org.eclipse.jdt.core.dom.BlockComment;
import org.eclipse.jdt.core.dom.BodyDeclaration;
import org.eclipse.jdt.core.dom.CatchClause;
import org.eclipse.jdt.core.dom.ClassInstanceCreation;
import org.eclipse.jdt.core.dom.Comment;
import org.eclipse.jdt.core.dom.CompilationUnit;
import org.eclipse.jdt.core.dom.DoStatement;
import org.eclipse.jdt.core.dom.EmptyStatement;
import org.eclipse.jdt.core.dom.EnhancedForStatement;
import org.eclipse.jdt.core.dom.Expression;
import org.eclipse.jdt.core.dom.ExpressionStatement;
import org.eclipse.jdt.core.dom.FieldDeclaration;
import org.eclipse.jdt.core.dom.ForStatement;
import org.eclipse.jdt.core.dom.IfStatement;
import org.eclipse.jdt.core.dom.InfixExpression;
import org.eclipse.jdt.core.dom.LabeledStatement;
import org.eclipse.jdt.core.dom.LineComment;
import org.eclipse.jdt.core.dom.MemberValuePair;
import org.eclipse.jdt.core.dom.MethodDeclaration;
import org.eclipse.jdt.core.dom.MethodInvocation;
import org.eclipse.jdt.core.dom.Modifier;
import org.eclipse.jdt.core.dom.NormalAnnotation;
import org.eclipse.jdt.core.dom.NumberLiteral;
import org.eclipse.jdt.core.dom.PrimitiveType;
import org.eclipse.jdt.core.dom.QualifiedName;
import org.eclipse.jdt.core.dom.ReturnStatement;
```

```
import org.eclipse.jdt.core.dom.SimpleName;
import org.eclipse.jdt.core.dom.SingleVariableDeclaration;
import org.eclipse.jdt.core.dom.Statement;
import org.eclipse.jdt.core.dom.StringLiteral;
import org.eclipse.jdt.core.dom.SwitchCase;
import org.eclipse.jdt.core.dom.SwitchStatement;
import org.eclipse.jdt.core.dom.TagElement;
import org.eclipse.jdt.core.dom.TextElement;
import org.eclipse.jdt.core.dom.ThrowStatement;
import org.eclipse.jdt.core.dom.TryStatement;
import org.eclipse.jdt.core.dom.TypeDeclaration;
import org.eclipse.jdt.core.dom.TypeDeclarationStatement;
import org.eclipse.jdt.core.dom.VariableDeclaration;
import org.eclipse.jdt.core.dom.VariableDeclarationExpression;
import org.eclipse.jdt.core.dom.VariableDeclarationFragment;
import org.eclipse.jdt.core.dom.VariableDeclarationStatement;
import org.eclipse.jdt.core.dom.WhileStatement;
import org.eclipse.jdt.core.dom.Assignment.Operator;
import org.eclipse.jdt.core.dom.rewrite.ASTRewrite;
import org.eclipse.jdt.core.dom.rewrite.ListRewrite;
import org.eclipse.jdt.core.dom.PostfixExpression;
import org.eclipse.jdt.internal.compiler.ast.CaseStatement;
import org.eclipse.jdt.internal.compiler.lookup.BlockScope;
import org.eclipse.jdt.internal.core.dom.rewrite.LineCommentEndOffsets;
import org.eclipse.jface.text.BadLocationException;
import org.eclipse.jface.text.Document;
import org.eclipse.jface.text.IDocument;
import org.eclipse.text.edits.MalformedTreeException;
import org.eclipse.text.edits.ReplaceEdit;
import org.eclipse.text.edits.TextEdit;
import org.omg.CORBA.INTF_REPOS;

public class CodeCounterSystem {
   // Keep block code.
   private Block[] blockArr = new Block[1];
   // Keep index line in block.
   private int[] indInnerBlock = new int[1];
   private int indBlock = 0;
   private String[] indOfEmptyBlock = new String[1];
   private Block[] blockEmpty = new Block[1];

   private String lineWithVar = "";
   private ArrayList<Integer> arLstLineNum = new ArrayList<Integer>();

   private String strLineNumLoopBlock = "";

   private int numOfCount = -1;
   private ASTRewrite astRewrite;
   private ArrayList<Object[]> arLstBlockLoop = new ArrayList<Object[]>();

   public void CodeCounter(String javaFilePath, String outputMapPathFolder)
{
      String path = javaFilePath;
      int intChkFirst = 0;

      try {
         //Convert java code to AST.
         File file = new File(path);
         BufferedReader in = new BufferedReader(new FileReader(file));
         StringBuffer buffer = new StringBuffer();
         String line = null;
         while ((line = in.readLine()) != null) {
            buffer.append("\t" + line);
            buffer.append("\n");
         }

         ASTParser parser = ASTParser.newParser(AST.JLS3);
         parser.setKind(ASTParser.K_COMPILATION_UNIT);
```

```
         String text = buffer.toString();
         Document doc = new Document(text);
         parser.setSource(doc.get().toCharArray());
         CompilationUnit node = (CompilationUnit) parser.createAST(null);
         AST ast = node.getAST();
         astRewrite = ASTRewrite.create(ast);

         node.recordModifications();

         //Class counter.
         int countClass = 0;
         for (Iterator iTerator = node.types().iterator(); iTerator
               .hasNext();) {
            Object stateIterate = iTerator.next();
            countClass++;
         }

         //Traverse AST node.
         TypeDeclaration p_typeDeclaration = null;
         for (int ind = 0; ind < countClass; ind++) {
            blockArr = new Block[1];
            indInnerBlock = new int[1];
            indBlock = 0;
            numOfCount = -1;
            lineWithVar = "";
            arLstLineNum = new ArrayList<Integer>();

            intChkFirst++;

            p_typeDeclaration = (TypeDeclaration) node.types().get(ind);
            lineWithVar = p_typeDeclaration.getName() + "\n";
            MethodDeclaration[] metDecArr = p_typeDeclaration.getMethods();

            for (int indMet = 0; indMet < metDecArr.length; indMet++) {
               //Loop for check profile annotation modifier for delete old
countNo and countNames.
               for (Iterator iMethodMod =
metDecArr[indMet].modifiers().iterator(); iMethodMod.hasNext();) {
                  //Object method modifier.
                  Object metModifier = iMethodMod.next();
                  //Find NormalAnnotation from each modifier.
                  if (metModifier instanceof NormalAnnotation) {
                     NormalAnnotation anno = (NormalAnnotation) metModifier;
                     if (anno.getTypeName().toString().equals("Profile")) {
                        for (Iterator iAno = anno.values().iterator();
iAno.hasNext();) {
                           Object annoProp = iAno.next();
                           MemberValuePair memValue = (MemberValuePair)
annoProp;
                           String valName = memValue.getName().toString();
                           if (valName.equals("countNo")) {
                              astRewrite.remove(memValue, null);
                           }
                           else if (valName.equals("countNames")) {
                              astRewrite.remove(memValue, null);
                           }
                        }
                     }
                  }
               }

               //First traverse, the data collection.
               GetLineNumber(metDecArr[indMet].getBody(), node);
               //Second traverse, modify and insert block into block array.
               TraverseForCount(metDecArr[indMet].getBody(), node);
            }

            //Insert countNo and countNames annotation.
```

```
            Boolean isProfile = false;
            Boolean isCountNo = false;
            Boolean isCountName = false;

            for (Iterator iTypeMod =
p_typeDeclaration.modifiers().iterator(); iTypeMod.hasNext();) {
               Object typeIterate = iTypeMod.next();
               if (typeIterate instanceof NormalAnnotation) {
                  NormalAnnotation anno = (NormalAnnotation) typeIterate;
                  if (anno.getTypeName().toString().equals("Profile")) {
                     isProfile = true;
                     for (Iterator iAno = anno.values().iterator();
iAno.hasNext();) {
                        Object annoProp = iAno.next();
                        MemberValuePair memValue = (MemberValuePair)
annoProp;
                        String valName = memValue.getName().toString();
                        if (valName.equals("countNo")) {
                           isCountNo = true;
                           NumberLiteral numLiteral =
ast.newNumberLiteral();
                           numLiteral.setToken(Integer.toString(numOfCount +
1));
                           astRewrite.set(memValue,
MemberValuePair.VALUE_PROPERTY, numLiteral, null);
                        }
                        else if (valName.equals("countNames")) {
                           isCountName = true;
                           ArrayInitializer arIni =
ast.newArrayInitializer();
                           StringLiteral strLite = ast.newStringLiteral();
                           for (int iLine = 0; iLine < arLstLineNum.size();
iLine++)
                           {
                              strLite = ast.newStringLiteral();
                              strLite.setEscapedValue("\"" +
Integer.toString(arLstLineNum.get(iLine)) + "\"");
                              arIni.expressions().add(strLite);
                           }
                           astRewrite.set(memValue,
MemberValuePair.VALUE_PROPERTY, arIni, null);
                        }
                     }
                     if (!isCountNo)
                     {
                        MemberValuePair countMem = ast.newMemberValuePair();
                        SimpleName simName = ast.newSimpleName("countNo");
                        countMem.setName(simName);
                        NumberLiteral numLiteral = ast.newNumberLiteral();
                        numLiteral.setToken(Integer.toString(numOfCount +
1));
                        countMem.setValue(numLiteral);

                        listRewrite= astRewrite.getListRewrite(anno,
NormalAnnotation.VALUES_PROPERTY);
                        listRewrite.insertLast(countMem, null);
                     }
                     if (!isCountName)
                     {
                        MemberValuePair countMem = ast.newMemberValuePair();
                        SimpleName simName =
ast.newSimpleName("countNames");
                        countMem.setName(simName);
                        ArrayInitializer arIni = ast.newArrayInitializer();
                        StringLiteral strLite = ast.newStringLiteral();
                        for (int iLine = 0; iLine < arLstLineNum.size();
iLine++)
                        {
```

```
                           strLite = ast.newStringLiteral();
                           strLite.setEscapedValue("\"" +
Integer.toString(arLstLineNum.get(iLine)) + "\"");
                           arIni.expressions().add(strLite);
                        }
                        countMem.setValue(arIni);

                        ListRewrite listRewrite=
astRewrite.getListRewrite(anno, NormalAnnotation.VALUES_PROPERTY);
                        listRewrite.insertLast(countMem, null);
                     }
                  }
               }
            }
            if (!isProfile)
            {
               NormalAnnotation anno = ast.newNormalAnnotation();
               SimpleName simName = ast.newSimpleName("Profile");
               anno.setTypeName(simName);

               MemberValuePair countMem = ast.newMemberValuePair();
               SimpleName simName2 = ast.newSimpleName("countNo");
               countMem.setName(simName2);
               NumberLiteral numLiteral = ast.newNumberLiteral();
               numLiteral.setToken(Integer.toString(numOfCount + 1));
               countMem.setValue(numLiteral);
               anno.values().add(countMem);

               countMem = ast.newMemberValuePair();
               SimpleName simCountName = ast.newSimpleName("countNames");
               countMem.setName(simCountName);
               ArrayInitializer arIni = ast.newArrayInitializer();
               StringLiteral strLite = ast.newStringLiteral();
               for (int iLine = 0; iLine < arLstLineNum.size(); iLine++)
               {
                  strLite = ast.newStringLiteral();
                  strLite.setEscapedValue("\"" +
Integer.toString(arLstLineNum.get(iLine)) + "\"");
                  arIni.expressions().add(strLite);
               }
               countMem.setValue(arIni);
               anno.values().add(countMem);

               ListRewrite listRewrite=
astRewrite.getListRewrite(p_typeDeclaration,
TypeDeclaration.MODIFIERS2_PROPERTY);
               listRewrite.insertFirst(anno, null);
            }

            if (blockArr[0] != null) {


               //Loop back array for insert count operation.
               for (int j = (blockArr.length - 1); j >= 0; j--) {
                  Block blNode = blockArr[j];
                  boolean isEmptyBlock = false;
                  if (indOfEmptyBlock[0] != null) {
                     for (int indEmp = 0; indEmp < indOfEmptyBlock.length;
indEmp++) {
                        String[] str =
indOfEmptyBlock[indEmp].toString().split(",");
                        int ind1 = Integer.parseInt(str[0]);
                        int ind2 = Integer.parseInt(str[1]);
                        if (ind1 == j) {
                           if (ind2 == indInnerBlock[j]) {
                              ListRewrite listRewrite=
astRewrite.getListRewrite(blockEmpty[indEmp], Block.STATEMENTS_PROPERTY);
                              ASTNode placeHolder=
```

```
astRewrite.createStringPlaceholder("//@Profiler.counts[" +
                                    Integer.toString(j) + "]++;",
ASTNode.RETURN_STATEMENT);
                                listRewrite.insertFirst(placeHolder, null);

                                isEmptyBlock = true;
                                break;
                             }
                          }
                       }
                    }
                    if (isEmptyBlock)
                       continue;

                    ListRewrite listRewrite= astRewrite.getListRewrite(blNode,
Block.STATEMENTS_PROPERTY);
                    ASTNode placeHolder=
astRewrite.createStringPlaceholder("//@Profiler.counts[" +
                    Integer.toString(j) + "]++;", ASTNode.RETURN_STATEMENT);
                    listRewrite.insertAt(placeHolder, indInnerBlock[j], null);
                 }
              }

              for (int iBL = 0; iBL < arLstBlockLoop.size(); iBL++)
              {
                 Object[] objAr = arLstBlockLoop.get(iBL);
                    Block blLoop = (Block)objAr[0];
                    ListRewrite listRewrite= astRewrite.getListRewrite(blLoop,
Block.STATEMENTS_PROPERTY);
                    ASTNode placeHolder=
astRewrite.createStringPlaceholder("//@Profiler.counts[" +
                    Integer.toString(Integer.parseInt(objAr[1].toString())) +
"]++;", ASTNode.RETURN_STATEMENT);
                    listRewrite.insertFirst(placeHolder, null);
              }

              //Write mapping line.
              String strCurrentPath = outputMapPathFolder;
              String[] pathCut = path.split("\\\\");
              String[] strFileName = pathCut[pathCut.length - 1].split("\\.");
              File dir = new File(strCurrentPath);
              BufferedWriter out = null;
              if (intChkFirst == 1) {
                 out = new BufferedWriter(new FileWriter(dir + "\\Map_"
                       + strFileName[0] + ".txt"));
                                    String firstLine = "Line, Variable" +
System.getProperty("line.separator");
                                    out.write(firstLine);
                              } else if (intChkFirst > 1) {
                                    out = new BufferedWriter(new FileWriter(dir
+ "\\Map_"
                                                + strFileName[0] + ".txt",
true));
              }

              lineWithVar = lineWithVar.replaceAll("\n",
System.getProperty("line.separator"));
              out.write(lineWithVar);
              out.flush();
              out.close();
           }

           TextEdit edits = astRewrite.rewriteAST(doc, null);
           edits.apply(doc);

           try {
              String strCurrentPath = path;
              String[] pathCut = path.split("\\\\");
```

```
            File dir = new File(strCurrentPath);
            if (!dir.exists()) {
               dir.mkdir();
            }

            //Write result to .java file.
            BufferedWriter out = new BufferedWriter(new FileWriter(dir));
            String getDoc = doc.get();
            getDoc = getDoc.replaceAll("\n",
System.getProperty("line.separator"));
            out.write(getDoc);
            out.flush();
            out.close();

            //Start and finish line of loop.
            String[] pathCutStr = path.split("\\\\");
            String[] strFileName = pathCutStr[pathCutStr.length -
1].split("\\.");
            File dirLoop = new File(outputMapPathFolder);

            BufferedWriter outBuffer = new BufferedWriter(new
FileWriter(dirLoop + "\\Loop_"
                     + strFileName[0] + ".txt"));

            outBuffer.write(strLineNumLoopBlock);
            outBuffer.flush();
            outBuffer.close();

         } catch (IOException e2) {
            // TODO Auto-generated catch block
            e2.printStackTrace();
         }
      } catch (IOException e) {
         try {
            throw new InvocationTargetException(e);
         } catch (InvocationTargetException e1) {
            // TODO Auto-generated catch block
            e1.printStackTrace();
         }
      } catch (MalformedTreeException e) {
         // TODO Auto-generated catch block
         e.printStackTrace();
      } catch (Exception e) {// BadLocationException e) {
         // TODO Auto-generated catch block
         e.printStackTrace();
      }
   }

   public CodeCounterSystem() {

   }

   private void GetLineFromIfElse(IfStatement ifState, CompilationUnit
comNode) {
      Block boxOfState = null;
      Object stateIterate = null;
      try {
         boxOfState = (Block) ifState.getThenStatement();
         if (boxOfState.statements().isEmpty()) {
            numOfCount++;
            lineWithVar += Integer.toString(comNode.getLineNumber(ifState
                  .getThenStatement().getStartPosition()))
                  + ", @Profiler.counts[" + Integer.toString(numOfCount) +
"]\n";

arLstLineNum.add(comNode.getLineNumber(ifState.getThenStatement().getStartPo
sition()));
         } else
```

```
            GetLineNumber(boxOfState, comNode);
      } catch (Exception ex) {
         numOfCount++;
         lineWithVar += Integer.toString(comNode.getLineNumber(ifState
               .getThenStatement().getStartPosition()))
               + ", @Profiler.counts[" + Integer.toString(numOfCount) +
"]\n";

arLstLineNum.add(comNode.getLineNumber(ifState.getThenStatement().getStartPo
sition()));
      }

      if (ifState.getElseStatement() != null) {
         try {
            stateIterate = ifState.getElseStatement();
            boxOfState = (Block) stateIterate;
            if (boxOfState.statements().isEmpty()) {
               numOfCount++;
               lineWithVar += Integer.toString(comNode
                     .getLineNumber(ifState.getElseStatement()
                           .getStartPosition()))
                     + ", @Profiler.counts[" + Integer.toString(numOfCount)
+ "]\n";

arLstLineNum.add(comNode.getLineNumber(ifState.getElseStatement().getStartPo
sition()));
            } else
               GetLineNumber(boxOfState, comNode);
         } catch (Exception ex) {
            if (stateIterate instanceof IfStatement)
               GetLineFromIfElse((IfStatement) stateIterate, comNode);
            else {
               numOfCount++;
               lineWithVar += Integer.toString(comNode
                     .getLineNumber(ifState.getElseStatement()
                           .getStartPosition()))
                     + ", @Profiler.counts[" + Integer.toString(numOfCount)
+ "]\n";

arLstLineNum.add(comNode.getLineNumber(ifState.getElseStatement().getStartPo
sition()));
            }
         }
      }
   }

   private void GetLineFromObject(Object stateMent, CompilationUnit comNode)
{
      if (stateMent instanceof SwitchStatement) {
         SwitchStatement stateCast = (SwitchStatement) stateMent;
         for (Iterator iTerator2 = stateCast.statements().iterator();
iTerator2.hasNext();) {
            Object stateIterate2 = iTerator2.next();
            Statement stMnt = (Statement) stateIterate2;
            System.out.println(stateIterate2);
            if (!(stateIterate2 instanceof SwitchCase)
                  || stateIterate2 instanceof SwitchStatement) {
               GetLineFromObject(stateIterate2, comNode);
            }
         }
      } else if (stateMent instanceof ForStatement) {
         ForStatement stateCast = (ForStatement) stateMent;
         Block boxOfState = null;
         try {
            boxOfState = (Block) stateCast.getBody();
            if (boxOfState.statements().isEmpty()) {
               numOfCount++;
               lineWithVar += Integer.toString(comNode
```

```
                        .getLineNumber(stateCast.getBody()
                              .getStartPosition()))
                        + ", @Profiler.counts[" + Integer.toString(numOfCount)
+ "]\n";

                strLineNumLoopBlock +=
Integer.toString(comNode.getLineNumber(stateCast.getBody().getStartPosition(
))) + "," +
                comNode.getLineNumber(stateCast.getBody().getStartPosition()
+ stateCast.getBody().getLength()) +
                System.getProperty("line.separator");


arLstLineNum.add(comNode.getLineNumber(stateCast.getBody().getStartPosition(
)));
             } else
             {
                strLineNumLoopBlock +=
Integer.toString(comNode.getLineNumber(stateCast.getBody().getStartPosition(
))) + "," +
                comNode.getLineNumber(stateCast.getBody().getStartPosition()
+ stateCast.getBody().getLength()) +
                System.getProperty("line.separator");

                GetLineNumber(boxOfState, comNode);
             }
          } catch (Exception ex) {
             numOfCount++;
             lineWithVar += Integer.toString(comNode.getLineNumber(stateCast
                   .getBody().getStartPosition()))
                   + ", @Profiler.counts[" + Integer.toString(numOfCount) +
"]\n";

             strLineNumLoopBlock +=
Integer.toString(comNode.getLineNumber(stateCast.getBody().getStartPosition(
))) + "," +
             comNode.getLineNumber(stateCast.getBody().getStartPosition() +
stateCast.getBody().getLength()) +
             System.getProperty("line.separator");


arLstLineNum.add(comNode.getLineNumber(stateCast.getBody().getStartPosition(
)));
          }
       } else if (stateMent instanceof TryStatement) {
          TryStatement stateCast = (TryStatement) stateMent;
          Block boxOfState = null;
          try {
             boxOfState = (Block) stateCast.getBody();
             if (boxOfState.statements().isEmpty()) {
                numOfCount++;
                lineWithVar += Integer.toString(comNode
                      .getLineNumber(stateCast.getBody()
                            .getStartPosition()))
                      + ", @Profiler.counts[" + Integer.toString(numOfCount)
+ "]\n";

arLstLineNum.add(comNode.getLineNumber(stateCast.getBody().getStartPosition(
)));
             } else
                GetLineNumber(boxOfState, comNode);
          } catch (Exception ex) {
             numOfCount++;
             lineWithVar += Integer.toString(comNode.getLineNumber(stateCast
                   .getBody().getStartPosition()))
                   + ", @Profiler.counts[" + Integer.toString(numOfCount) +
"]\n";
```

```
arLstLineNum.add(comNode.getLineNumber(stateCast.getBody().getStartPosition(
)));
         }

         for (Iterator itCatch = stateCast.catchClauses().iterator();
itCatch.hasNext();) {
            Object itCatchObj = itCatch.next();
            CatchClause trState = (CatchClause) itCatchObj;
            stateMent = trState.getBody();
            try {
               boxOfState = (Block) stateMent;
               if (boxOfState.statements().isEmpty()) {
                  numOfCount++;
                  lineWithVar += Integer.toString(comNode
                        .getLineNumber(trState.getBody()
                              .getStartPosition()))
                        + ", @Profiler.counts["
                        + Integer.toString(numOfCount)
                        + "]\n";

arLstLineNum.add(comNode.getLineNumber(trState.getBody().getStartPosition())
);
               } else
                  GetLineNumber(boxOfState, comNode);
            } catch (Exception ex) {
               numOfCount++;
               lineWithVar += Integer.toString(comNode
                     .getLineNumber(trState.getStartPosition()))
                     + ", @Profiler.counts[" + Integer.toString(numOfCount)
+ "]\n";

arLstLineNum.add(comNode.getLineNumber(trState.getStartPosition()));
            }
         }
      } else if (stateMent instanceof DoStatement) {
         DoStatement stateCast = (DoStatement) stateMent;
         Block boxOfState = null;
         try {
            boxOfState = (Block) stateCast.getBody();
            if (boxOfState.statements().isEmpty()) {
               numOfCount++;
               lineWithVar +=
Integer.toString(comNode.getLineNumber(stateCast.getBody().getStartPosition(
)))
                     + ", @Profiler.counts[" + Integer.toString(numOfCount)
+ "]\n";

arLstLineNum.add(comNode.getLineNumber(stateCast.getBody().getStartPosition(
)));

               strLineNumLoopBlock +=
Integer.toString(comNode.getLineNumber(stateCast.getBody().getStartPosition(
))) + "," +
               comNode.getLineNumber(stateCast.getBody().getStartPosition()
+ stateCast.getBody().getLength()) +
               System.getProperty("line.separator");
            } else
            {
               strLineNumLoopBlock +=
Integer.toString(comNode.getLineNumber(stateCast.getBody().getStartPosition(
))) + "," +
               comNode.getLineNumber(stateCast.getBody().getStartPosition()
+ stateCast.getBody().getLength()) +
               System.getProperty("line.separator");

               GetLineNumber(boxOfState, comNode);
            }
         } catch (Exception ex) {
```

```
            numOfCount++;
            lineWithVar += Integer.toString(comNode.getLineNumber(stateCast
                  .getBody().getStartPosition()))
                  + ", @Profiler.counts[" + Integer.toString(numOfCount) +
"]\n";

arLstLineNum.add(comNode.getLineNumber(stateCast.getBody().getStartPosition(
)));

            strLineNumLoopBlock +=
Integer.toString(comNode.getLineNumber(stateCast.getBody().getStartPosition(
)) + "," +
            comNode.getLineNumber(stateCast.getBody().getStartPosition() +
stateCast.getBody().getLength()) +
            System.getProperty("line.separator");
         }
      } else if (stateMent instanceof EnhancedForStatement) {
         EnhancedForStatement stateCast = (EnhancedForStatement) stateMent;
         Block boxOfState = null;
         try {
            boxOfState = (Block) stateCast.getBody();
            if (boxOfState.statements().isEmpty()) {
               numOfCount++;
               lineWithVar += Integer.toString(comNode
                     .getLineNumber(boxOfState.getStartPosition()))
                     + ", @Profiler.counts[" + Integer.toString(numOfCount)
+ "]\n";

arLstLineNum.add(comNode.getLineNumber(boxOfState.getStartPosition()));

               strLineNumLoopBlock +=
Integer.toString(comNode.getLineNumber(boxOfState.getStartPosition())) + ","
+ comNode.getLineNumber(boxOfState.getStartPosition() +
boxOfState.getLength()) +
               System.getProperty("line.separator");
            } else
            {
               strLineNumLoopBlock +=
Integer.toString(comNode.getLineNumber(boxOfState.getStartPosition())) + ","
+
               comNode.getLineNumber(boxOfState.getStartPosition() +
boxOfState.getLength()) +
               System.getProperty("line.separator");

               GetLineNumber(boxOfState, comNode);
            }
         } catch (Exception ex) {
            numOfCount++;
            lineWithVar += Integer.toString(comNode.getLineNumber(stateCast
                  .getBody().getStartPosition()))
                  + ", @Profiler.counts[" + Integer.toString(numOfCount) +
"]\n";

arLstLineNum.add(comNode.getLineNumber(stateCast.getBody().getStartPosition(
)));

            strLineNumLoopBlock +=
Integer.toString(comNode.getLineNumber(stateCast.getBody().getStartPosition(
)) + "," +
            comNode.getLineNumber(stateCast.getBody().getStartPosition() +
stateCast.getBody().getLength()) +
            System.getProperty("line.separator");
         }
      } else if (stateMent instanceof IfStatement) {
         IfStatement stateCast = (IfStatement) stateMent;
         GetLineFromIfElse(stateCast, comNode);
      } else if (stateMent instanceof LabeledStatement) {
         LabeledStatement stateCast = (LabeledStatement) stateMent;
```

```
         Block boxOfState = null;
         try {
            boxOfState = (Block) stateCast.getBody();
            if (boxOfState.statements().isEmpty()) {
               numOfCount++;
               lineWithVar += Integer.toString(comNode
                     .getLineNumber(stateCast.getBody()
                           .getStartPosition()))
                     + ", @Profiler.counts[" + Integer.toString(numOfCount)
+ "]\n";

arLstLineNum.add(comNode.getLineNumber(stateCast.getBody().getStartPosition(
)));
            } else
               GetLineNumber(boxOfState, comNode);
         } catch (Exception ex) {
            numOfCount++;
            lineWithVar += Integer.toString(comNode.getLineNumber(stateCast
                  .getBody().getStartPosition()))
                  + ", @Profiler.counts[" + Integer.toString(numOfCount) +
"]\n";

arLstLineNum.add(comNode.getLineNumber(stateCast.getBody().getStartPosition(
)));
         }
      } else if (stateMent instanceof WhileStatement) {
         WhileStatement stateCast = (WhileStatement) stateMent;
         Block boxOfState = null;
         try {
            boxOfState = (Block) stateCast.getBody();
            if (boxOfState.statements().isEmpty()) {
               numOfCount++;
               lineWithVar += Integer.toString(comNode
                     .getLineNumber(stateCast.getBody()
                           .getStartPosition()))
                     + ", @Profiler.counts[" + Integer.toString(numOfCount)
+ "]\n";

arLstLineNum.add(comNode.getLineNumber(stateCast.getBody().getStartPosition(
)));

               strLineNumLoopBlock +=
Integer.toString(comNode.getLineNumber(stateCast.getBody().getStartPosition(
))) + "," +
               comNode.getLineNumber(stateCast.getBody().getStartPosition()
+ stateCast.getBody().getLength()) +
               System.getProperty("line.separator");
            } else
            {
               strLineNumLoopBlock +=
Integer.toString(comNode.getLineNumber(stateCast.getBody().getStartPosition(
))) + "," +
               comNode.getLineNumber(stateCast.getBody().getStartPosition()
+ stateCast.getBody().getLength()) +
               System.getProperty("line.separator");

               GetLineNumber(boxOfState, comNode);
            }
         } catch (Exception ex) {
            numOfCount++;
            lineWithVar += Integer.toString(comNode.getLineNumber(stateCast
                  .getBody().getStartPosition()))
                  + ", @Profiler.counts[" + Integer.toString(numOfCount) +
"]\n";

arLstLineNum.add(comNode.getLineNumber(stateCast.getBody().getStartPosition(
)));
```

```
            strLineNumLoopBlock +=
Integer.toString(comNode.getLineNumber(stateCast.getBody().getStartPosition(
))) + "," +
            comNode.getLineNumber(stateCast.getBody().getStartPosition() +
stateCast.getBody().getLength()) +
            System.getProperty("line.separator");
         }
      } else if (stateMent instanceof BlockComment) {
      }
      Statement stateTmp = (Statement) stateMent;
      numOfCount++;
      lineWithVar += Integer.toString(comNode.getLineNumber(stateTmp
            .getStartPosition()))
            + ", @Profiler.counts[" + Integer.toString(numOfCount) + "]\n";
      arLstLineNum.add(comNode.getLineNumber(stateTmp.getStartPosition()));
   }

   private void GetLineNumber(Block node, CompilationUnit comNode) {
      if (node != null) {
         if (!node.statements().isEmpty()) {
            for (Iterator iTerator = node.statements().iterator(); iTerator
                  .hasNext();) {
               Object stateIterate = iTerator.next();
               GetLineFromObject(stateIterate, comNode);
            }
         }
      }
   }

   private void TraverseForCountIfElse(IfStatement ifState,
         CompilationUnit comNode, Object stateIterate, int indOfBlock,
         Block node) {
      Block boxOfState = null;
      try {
         boxOfState = (Block) ifState.getThenStatement();
         if (boxOfState.statements().isEmpty())
            GetEmptyBlock(boxOfState, indOfBlock, node, comNode);
         else
            TraverseForCount(boxOfState, comNode);
      } catch (Exception ex) {
         if (ifState.getThenStatement() instanceof EmptyStatement) {
            AST ast = comNode.getAST();
            Block bl = ast.newBlock();

            astRewrite.set(ifState, IfStatement.THEN_STATEMENT_PROPERTY, bl,
null);

            ifState.setThenStatement(bl);
            boxOfState = (Block) ifState.getThenStatement();
            GetEmptyBlock(boxOfState, indOfBlock, node, comNode);
         } else {
            Statement Exp = ifState.getThenStatement();
            int line = comNode.getLineNumber(ifState.getStartPosition());
            AST ast = comNode.getAST();
            Block bl = ast.newBlock();

            astRewrite.set(ifState, IfStatement.THEN_STATEMENT_PROPERTY, bl,
null);
            ListRewrite listRewrite= astRewrite.getListRewrite(bl,
Block.STATEMENTS_PROPERTY);
            listRewrite.insertFirst(Exp, null);

            ifState.setThenStatement(bl);
            bl.statements().add(Exp);

            boxOfState = (Block) ifState.getThenStatement();
            TraverseForCount(boxOfState, comNode);
         }
```

```
      }

      if (ifState.getElseStatement() != null) {
         try {
            stateIterate = ifState.getElseStatement();
            boxOfState = (Block) stateIterate;
            if (boxOfState.statements().isEmpty())
               GetEmptyBlock(boxOfState, indOfBlock, node, comNode);
            else
               TraverseForCount(boxOfState, comNode);
         } catch (Exception ex) {
            if (stateIterate instanceof IfStatement)
               TraverseForCountIfElse((IfStatement) stateIterate, comNode,
                     stateIterate, indOfBlock, node);
            else {
               if (ifState.getElseStatement() instanceof EmptyStatement) {
                  AST ast = comNode.getAST();
                  Block bl = ast.newBlock();

                  astRewrite.set(ifState,
IfStatement.ELSE_STATEMENT_PROPERTY, bl, null);

                  ifState.setElseStatement(bl);
                  stateIterate = ifState.getElseStatement();
                  boxOfState = (Block) stateIterate;
                  GetEmptyBlock(boxOfState, indOfBlock, node, comNode);
               } else {
                  Statement Exp = ifState.getElseStatement();
                  int line = comNode.getLineNumber(ifState
                        .getStartPosition());
                  AST ast = comNode.getAST();
                  Block bl = ast.newBlock();

                  astRewrite.set(ifState,
IfStatement.ELSE_STATEMENT_PROPERTY, bl, null);
                  ListRewrite listRewrite= astRewrite.getListRewrite(bl,
Block.STATEMENTS_PROPERTY);
                  listRewrite.insertFirst(Exp, null);

                  ifState.setElseStatement(bl);
                  bl.statements().add(Exp);
                  boxOfState = (Block) ifState.getElseStatement();
                  TraverseForCount(boxOfState, comNode);
               }
            }
         }
      }
   }

   private void TraverseForCountObject(Object stateMent,
         CompilationUnit comNode, int indOfBlock, Block node) {
      AST ast = comNode.getAST();

      if (stateMent instanceof SwitchStatement) {
         SwitchStatement stateCast = (SwitchStatement) stateMent;
         int indInternal = -1;
         // int lineStParent =
         // comNode.getLineNumber(blState.getStartPosition());

         for (Iterator iTerator2 = stateCast.statements().iterator();
iTerator2.hasNext();) {
            Object stateIterate2 = iTerator2.next();
            indInternal++;
            Statement stMnt = (Statement) stateIterate2;
            if (!(stateIterate2 instanceof SwitchCase)
                  || stateIterate2 instanceof SwitchStatement) {
               Block bl = ast.newBlock();
               stateCast.statements().set(indInternal, bl);
```

```
                bl.statements().add(stMnt);
                TraverseForCount(bl, comNode);
             }
          }
       } else if (stateMent instanceof ForStatement) {
          ForStatement stateCast = (ForStatement) stateMent;
          Block boxOfState = null;
          try {
             boxOfState = (Block) stateCast.getBody();
             if (boxOfState.statements().isEmpty())
                GetEmptyBlock(boxOfState, indOfBlock, node, comNode);
             else
                TraverseForCount(boxOfState, comNode);
          } catch (Exception ex) {
             if (stateCast.getBody() instanceof EmptyStatement) {
                Block bl = ast.newBlock();
                astRewrite.set(stateCast, ForStatement.BODY_PROPERTY, bl,
null);

                stateCast.setBody(bl);
                boxOfState = (Block) stateCast.getBody();
                GetEmptyBlock(boxOfState, indOfBlock, node, comNode);
             } else {
                Statement Exp = stateCast.getBody();
                Block bl = ast.newBlock();

                astRewrite.set(stateCast, ForStatement.BODY_PROPERTY, bl,
null);
                ListRewrite listRewrite= astRewrite.getListRewrite(bl,
Block.STATEMENTS_PROPERTY);
                listRewrite.insertFirst(Exp, null);

                stateCast.setBody(bl);
                bl.statements().add(Exp);
                boxOfState = (Block) stateCast.getBody();
                TraverseForCount(boxOfState, comNode);
             }
          }
       } else if (stateMent instanceof TryStatement) {
          TryStatement stateCast = (TryStatement) stateMent;
          Block boxOfState = null;
          try {
             boxOfState = (Block) stateCast.getBody();
             if (boxOfState.statements().isEmpty())
                GetEmptyBlock(boxOfState, indOfBlock, node, comNode);
             else
                TraverseForCount(boxOfState, comNode);
          } catch (Exception ex) {
             Block bl = ast.newBlock();

             astRewrite.set(stateCast, TryStatement.BODY_PROPERTY, bl, null);

             stateCast.setBody(bl);
             boxOfState = (Block) stateCast.getBody();
             GetEmptyBlock(boxOfState, indOfBlock, node, comNode);
          }

          for (Iterator itCatch = stateCast.catchClauses().iterator();
itCatch.hasNext();) {
             Object itCatchObj = itCatch.next();
             CatchClause trState = (CatchClause) itCatchObj;
             stateMent = trState.getBody();
             try {
                boxOfState = (Block) stateMent;
                if (boxOfState.statements().isEmpty())
                   GetEmptyBlock(boxOfState, indOfBlock, node, comNode);
                else
                   TraverseForCount(boxOfState, comNode);
```

```
            } catch (Exception ex) {
               Block bl = ast.newBlock();

               astRewrite.set(trState, CatchClause.BODY_PROPERTY, bl, null);

               trState.setBody(bl);
               stateMent = trState.getBody();
               boxOfState = (Block) stateMent;
               GetEmptyBlock(boxOfState, indOfBlock, node, comNode);
            }
         }
      } else if (stateMent instanceof DoStatement) {
         DoStatement stateCast = (DoStatement) stateMent;
         Block boxOfState = null;
         try {
            boxOfState = (Block) stateCast.getBody();
            if (boxOfState.statements().isEmpty())
               GetEmptyBlock(boxOfState, indOfBlock, node, comNode);
            else
               TraverseForCount(boxOfState, comNode);
         } catch (Exception ex) {
            if (stateCast.getBody() instanceof EmptyStatement) {
               Block bl = ast.newBlock();

               astRewrite.set(stateCast, DoStatement.BODY_PROPERTY, bl,
null);

               stateCast.setBody(bl);
               boxOfState = (Block) stateCast.getBody();
               GetEmptyBlock(boxOfState, indOfBlock, node, comNode);
            } else {
               Statement Exp = stateCast.getBody();
               Block bl = ast.newBlock();

               astRewrite.set(stateCast, DoStatement.BODY_PROPERTY, bl,
null);
               ListRewrite listRewrite= astRewrite.getListRewrite(bl,
Block.STATEMENTS_PROPERTY);
               listRewrite.insertFirst(Exp, null);

               stateCast.setBody(bl);
               bl.statements().add(Exp);
               boxOfState = (Block) stateCast.getBody();
               TraverseForCount(boxOfState, comNode);
            }
         }
      } else if (stateMent instanceof EnhancedForStatement) {
         EnhancedForStatement stateCast = (EnhancedForStatement) stateMent;
         Block boxOfState = null;
         try {
            boxOfState = (Block) stateCast.getBody();
            if (boxOfState.statements().isEmpty())
               GetEmptyBlock(boxOfState, indOfBlock, node, comNode);
            else
               TraverseForCount(boxOfState, comNode);
         } catch (Exception ex) {
            if (stateCast.getBody() instanceof EmptyStatement) {
               Block bl = ast.newBlock();
               astRewrite.set(stateCast, EnhancedForStatement.BODY_PROPERTY,
bl, null);

               stateCast.setBody(bl);
               boxOfState = (Block) stateCast.getBody();
               GetEmptyBlock(boxOfState, indOfBlock, node, comNode);
            } else {
               Statement Exp = stateCast.getBody();
               Block bl = ast.newBlock();
```

```
                astRewrite.set(stateCast, EnhancedForStatement.BODY_PROPERTY,
bl, null);
                ListRewrite listRewrite= astRewrite.getListRewrite(bl,
Block.STATEMENTS_PROPERTY);
                listRewrite.insertFirst(Exp, null);

                stateCast.setBody(bl);
                bl.statements().add(Exp);
                boxOfState = (Block) stateCast.getBody();
                TraverseForCount(boxOfState, comNode);
             }
          }
       } else if (stateMent instanceof IfStatement) {
          IfStatement stateCast = (IfStatement) stateMent;
          TraverseForCountIfElse(stateCast, comNode, stateMent, indOfBlock,
                node);
       } else if (stateMent instanceof LabeledStatement) {
          LabeledStatement stateCast = (LabeledStatement) stateMent;
          Block boxOfState = null;
          try {
             boxOfState = (Block) stateCast.getBody();
             if (boxOfState.statements().isEmpty())
                GetEmptyBlock(boxOfState, indOfBlock, node, comNode);
             else
                TraverseForCount(boxOfState, comNode);
          } catch (Exception ex) {
             if (stateCast.getBody() instanceof EmptyStatement) {
                Block bl = ast.newBlock();

                astRewrite.set(stateCast, LabeledStatement.BODY_PROPERTY, bl,
null);

                stateCast.setBody(bl);
                boxOfState = (Block) stateCast.getBody();
                GetEmptyBlock(boxOfState, indOfBlock, node, comNode);
             } else {
                Statement Exp = stateCast.getBody();
                Block bl = ast.newBlock();

                astRewrite.set(stateCast, LabeledStatement.BODY_PROPERTY, bl,
null);
                ListRewrite listRewrite= astRewrite.getListRewrite(bl,
Block.STATEMENTS_PROPERTY);
                listRewrite.insertFirst(Exp, null);

                stateCast.setBody(bl);
                bl.statements().add(Exp);
                boxOfState = (Block) stateCast.getBody();
                TraverseForCount(boxOfState, comNode);
             }
          }
       } else if (stateMent instanceof WhileStatement) {
          WhileStatement stateCast = (WhileStatement) stateMent;
          Block boxOfState = null;
          try {
             boxOfState = (Block) stateCast.getBody();
             if (boxOfState.statements().isEmpty())
                GetEmptyBlock(boxOfState, indOfBlock, node, comNode);
             else
                TraverseForCount(boxOfState, comNode);
          } catch (Exception ex) {
             if (stateCast.getBody() instanceof EmptyStatement) {
                Block bl = ast.newBlock();

                astRewrite.set(stateCast, WhileStatement.BODY_PROPERTY, bl,
null);

                stateCast.setBody(bl);
```

```
                boxOfState = (Block) stateCast.getBody();
                GetEmptyBlock(boxOfState, indOfBlock, node, comNode);
             } else {
                Statement Exp = stateCast.getBody();
                Block bl = ast.newBlock();

                astRewrite.set(stateCast, WhileStatement.BODY_PROPERTY, bl,
null);
                ListRewrite listRewrite= astRewrite.getListRewrite(bl,
Block.STATEMENTS_PROPERTY);
                listRewrite.insertFirst(Exp, null);

                stateCast.setBody(bl);
                bl.statements().add(Exp);
                boxOfState = (Block) stateCast.getBody();
                TraverseForCount(boxOfState, comNode);
             }
          }
       }

       //Keep loop in array for add counter.
       if (stateMent instanceof WhileStatement)
       {
          Object[] objAr = new Object[3];
          WhileStatement stateCast = (WhileStatement) stateMent;
          Block boxOfState = null;
          try {
             boxOfState = (Block) stateCast.getBody();
             objAr[0] = boxOfState;
          } catch (Exception ex) {
             if (stateCast.getBody() instanceof EmptyStatement) {
                Block bl = ast.newBlock();

                astRewrite.set(stateCast, WhileStatement.BODY_PROPERTY, bl,
null);

                stateCast.setBody(bl);
                boxOfState = (Block) stateCast.getBody();
                objAr[0] = boxOfState;
             } else {
                Statement Exp = stateCast.getBody();
                Block bl = ast.newBlock();

                astRewrite.set(stateCast, WhileStatement.BODY_PROPERTY, bl,
null);
                ListRewrite listRewrite= astRewrite.getListRewrite(bl,
Block.STATEMENTS_PROPERTY);
                listRewrite.insertFirst(Exp, null);

                stateCast.setBody(bl);
                bl.statements().add(Exp);
                boxOfState = (Block) stateCast.getBody();
                objAr[0] = boxOfState;
             }
          }

          objAr[1] = blockArr.length;
          arLstBlockLoop.add(objAr);
          GetCount(indOfBlock, node, comNode);
       }
       else if (stateMent instanceof ForStatement) {
          Object[] objAr = new Object[3];
          ForStatement stateCast = (ForStatement) stateMent;
          Block boxOfState = null;
          try {
             boxOfState = (Block) stateCast.getBody();
             objAr[0] = boxOfState;
          } catch (Exception ex) {
```

```
            if (stateCast.getBody() instanceof EmptyStatement) {
               Block bl = ast.newBlock();

               astRewrite.set(stateCast, ForStatement.BODY_PROPERTY, bl,
null);

               stateCast.setBody(bl);
               boxOfState = (Block) stateCast.getBody();
               objAr[0] = boxOfState;
            } else {
               Statement Exp = stateCast.getBody();
               Block bl = ast.newBlock();

               astRewrite.set(stateCast, ForStatement.BODY_PROPERTY, bl,
null);
               ListRewrite listRewrite= astRewrite.getListRewrite(bl,
Block.STATEMENTS_PROPERTY);
               listRewrite.insertFirst(Exp, null);

               stateCast.setBody(bl);
               bl.statements().add(Exp);
               boxOfState = (Block) stateCast.getBody();
               objAr[0] = boxOfState;
            }
         }
         objAr[1] = blockArr.length;
         arLstBlockLoop.add(objAr);
         GetCount(indOfBlock, node, comNode);
      }
      else
         GetCount(indOfBlock, node, comNode);
   }

   private void TraverseForCount(Block node, CompilationUnit comNode) {
      if (node != null) {
         if (!node.statements().isEmpty()) {
            for (Iterator iTerator = node.statements().iterator(); iTerator
                  .hasNext();) {
               indOfBlock++;
               Object stateIterate = iTerator.next();

               TraverseForCountObject(stateIterate, comNode, indOfBlock,
                     node);
            }
         }
      }
   }

   private void GetEmptyBlock(Block boxOfState, int indOfBlock, Block node,
         CompilationUnit comNode) {
      if (indOfEmptyBlock[0] == null) {
         indOfEmptyBlock[0] = Integer.toString(indBlock) + ","
               + Integer.toString(indOfBlock);
         blockEmpty[0] = boxOfState;
      } else {
         String[] strTmp = indOfEmptyBlock.clone();
         indOfEmptyBlock = new String[indOfEmptyBlock.length + 1];
         Block[] blTmp = blockEmpty.clone();
         blockEmpty = new Block[blockEmpty.length + 1];
         for (int i = 0; i < strTmp.length; i++) {
            indOfEmptyBlock[i] = strTmp[i];
            blockEmpty[i] = blTmp[i];
         }
         indOfEmptyBlock[indOfEmptyBlock.length - 1] = Integer
               .toString(indBlock)
               + "," + Integer.toString(indOfBlock);
         blockEmpty[blockEmpty.length - 1] = boxOfState;
      }
```

```
      GetCount(indOfBlock, node, comNode);
   }

   private void GetCount(int indOfBlock, Block node, CompilationUnit
comNode) {
      if (indBlock > 0) {
         Block[] blTmp = blockArr.clone();
         int[] indInnerTmp = indInnerBlock.clone();
         blockArr = new Block[indBlock + 1];
         indInnerBlock = new int[indBlock + 1];
         for (int i = 0; i < blTmp.length; i++) {
            blockArr[i] = blTmp[i];
            indInnerBlock[i] = indInnerTmp[i];
         }
         blockArr[indBlock] = node;
         indInnerBlock[indBlock] = indOfBlock;

         indBlock++;
      } else {
         blockArr[indBlock] = node;
         indInnerBlock[indBlock] = indOfBlock;

         indBlock++;
      }
   }

   /**
    * @param args
    */
   public static void main(String[] args) {
      // TODO Auto-generated method stub
      CodeCounterSystem CCS = new CodeCounterSystem();

   }
}
```

รหัสที่ ก.1 รหัสการแทรกคำสั่งนับด้วย AST

ภาคผนวก ข คำสั่งสร้างภาพฮิสโตแกรม

ข.1 รหัสคำสั่งการแปลข้อมูลผลลัพธ์การทดลองให้แสดงผลภาพฮิสโตแกรม

```
package CountImageGenerator;

import java.awt.Color;
import java.awt.Font;
import java.awt.Graphics2D;
import java.awt.Image;
import java.awt.RenderingHints;
import java.awt.font.FontRenderContext;
import java.awt.font.LineBreakMeasurer;
import java.awt.font.TextAttribute;
import java.awt.font.TextLayout;
import java.awt.geom.Rectangle2D;
import java.awt.image.BufferedImage;
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.OutputStream;
import java.text.AttributedCharacterIterator;
import java.text.AttributedString;
import java.util.ArrayList;
import java.util.Date;
import java.util.Properties;
import javax.imageio.ImageIO;
import org.omg.CORBA.Environment;

public class ImageGenerator {

   static ArrayList<BufferedImage> images;
   Image img;
   static int lineHeight;
   static int width = 1000;
   static int margin = 415;
   static String fontName = "Arial";
   static int fontSize = 14;
   static Date ts;
   Image i;

   private Integer[] GetLineCount(ArrayList<Integer[]> intArrayLineAndCount,
Integer lineNumber)
   {
      for (int iAr = 0; iAr < intArrayLineAndCount.size(); iAr++)
      {
         Integer[] lineCount = intArrayLineAndCount.get(iAr);
         if(lineCount[0] == lineNumber)
            return lineCount;
      }
      return null;
   }

   /**
    *
    * @param intArrayLineAndCount   contain Integer[2] (line number, line
counter).
    * @param focusLine   the line for show image. (ex "12-24")
```

```
    */
   public void CounterToImg(String javaFilePath,String imgFolderPath,
         ArrayList<Integer[]> intArrayLineAndCount, Integer maxCount, String
strFileName, String focusLine)
   {
      String[] strCut = javaFilePath.split("\\\\");
      String fileName = strCut[strCut.length - 1];
      String filePath = "";
      ArrayList<String[]> arrStrLineAndCount = new ArrayList<String[]>();
      for (int iPth = 0; iPth < (strCut.length - 1); iPth++)
      {
         filePath += strCut[iPth] + "\\";
      }
      strCut = fileName.split("\\.");
      fileName = strCut[0];
      fileName += "_" + strFileName;

      String line = "";
      Integer lineCounter = 1;
      Integer lineNumber = 1;
      Integer imgCounter = 0;

      Integer minLine = 0;
      Integer maxLine = 0;
      if (focusLine != "")
      {
         String[] focusCut = focusLine.split("\\-");
         minLine = Integer.parseInt(focusCut[0]);
         maxLine = Integer.parseInt(focusCut[1]);
      }
      try
      {
         File file = new File(javaFilePath);
         BufferedReader in = new BufferedReader(new FileReader(file));
         while ((line = in.readLine()) != null) {
               String CodeLine = lineNumber.toString() + "\t" + line + "\n";
               if (focusLine != "")
               {
                  if (lineNumber < minLine)
                  {
                     lineNumber++;
                     continue;
                  }
                  else if (lineNumber > maxLine)
                     break;
               }
               Integer[] lineCount = GetLineCount(intArrayLineAndCount,
lineNumber);
               String[] lineAndCount = new String[3];
               if (lineCount != null)
               {
                  lineAndCount[0] = CodeLine;
                  double max = maxCount;
                  double intCount = lineCount[1];
                  double perCount = (intCount * 100) / max;
                  int pCount = (int)perCount;
                  lineAndCount[1] = Integer.toString(pCount);
                  lineAndCount[2] = Integer.toString(lineCount[1]);
               }
               else
               {
                  lineAndCount[0] = CodeLine;
                  lineAndCount[1] = "0";
                  lineAndCount[2] = "0";
               }
               arrStrLineAndCount.add(lineAndCount);

               //If line = 60 send to gen img.
```

```
               if (lineCounter == 60)
               {
                  imgCounter++;
                  ConvertCount(imgFolderPath + "\\" + fileName + "_" +
Integer.toString(imgCounter) + ".jpg", arrStrLineAndCount);
                  lineCounter = 0;
                  arrStrLineAndCount = new ArrayList<String[]>();
               }
               lineCounter++;
               lineNumber++;
         }
         if (arrStrLineAndCount.size() > 0)
         {
            imgCounter++;
            ConvertCount(imgFolderPath + "\\" + fileName + "_" +
Integer.toString(imgCounter) + ".jpg", arrStrLineAndCount);
         }
      }
      catch (Exception ex)
      { }
   }

   //ArrayList line and count contain String[3] (Code string of each line,
Count percent, Count number)
   private void ConvertCount(String imgFilePath, ArrayList<String[]>
arrayLineAndCount)
   {
      ts = new Date((new Date()).getTime() - (24 * 60 * 60 * 1000));
      Graphics2D g, g1;
      FontRenderContext fc;
      String testText = "Testing";
      int fileCnt = 0;

      Font font = null;

      try {
         font = new Font(fontName, Font.PLAIN, (int) fontSize);
      } catch (Exception e) {
         System.out.println("Error opening font file : " + e.getMessage());
      }

      BufferedImage tempBuffer = new BufferedImage(1, 1,
BufferedImage.TYPE_INT_RGB);
      g = tempBuffer.createGraphics();
      g.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
RenderingHints.VALUE_ANTIALIAS_ON);
      g.setColor(Color.BLACK);
      g.setBackground(Color.WHITE);
      g.setFont(font);
      fc = g.getFontRenderContext();

      Rectangle2D bounds = font.getStringBounds(testText, fc);
      float wrappingWidth = width;
      lineHeight = (int) bounds.getHeight();

      images = new ArrayList<BufferedImage>();
      int lineCnt = 10;//margin + margin;

      String line = "";

      for (int iLst = 0; iLst < arrayLineAndCount.size(); iLst++) {
         String[] strLine = arrayLineAndCount.get(iLst);
         line = strLine[0].toString();

         String line2 = "";
         String sBkground =
                           "
" +
```

```
                              "
" +
                              "
";
         Integer countPer = Integer.parseInt(strLine[1].toString());
         Integer countNum = Integer.parseInt(strLine[2].toString());

         while (countPer > 0)
         {
            line2 += "l";
            countPer--;
         }

         while (line.length() < 500)
            line += " ";

         //Code
         AttributedString attribString = new AttributedString(line);
         attribString.addAttribute(TextAttribute.FOREGROUND, Color.BLACK, 0,
line.length());
         attribString.addAttribute(TextAttribute.BACKGROUND, Color.WHITE, 0,
line.length());
         attribString.addAttribute(TextAttribute.FONT, font, 0,
line.length());
         AttributedCharacterIterator aci = attribString.getIterator();
         LineBreakMeasurer lbm = new LineBreakMeasurer(aci, fc);

         //Histogram
         AttributedString attribString2 = new AttributedString(line2);
         LineBreakMeasurer lbm2 = null;

         if (line2.length() > 0)
         {
            Font font2 = null;
            try {
               font2 = new Font(fontName, Font.PLAIN, ((int) fontSize)-2);
            } catch (Exception e) {
               System.out.println("Error opening font file : " +
e.getMessage());
            }

            attribString2.addAttribute(TextAttribute.FOREGROUND,
Color.DARK_GRAY, 0,   line2.length());
            attribString2.addAttribute(TextAttribute.BACKGROUND,
Color.DARK_GRAY, 0,   line2.length());
            attribString2.addAttribute(TextAttribute.FONT, font2, 0,
line2.length());
            AttributedCharacterIterator aci2 = attribString2.getIterator();
            lbm2 = new LineBreakMeasurer(aci2, fc);
         }

         //Line Counter
         String sCntNum = Integer.toString(countNum);
         AttributedString attCountNum = new AttributedString(sCntNum);
         LineBreakMeasurer lbmCntNum = null;
         if (countNum > 0)
         {
            Font fontCntNum = null;
            try {
               fontCntNum = new Font(fontName, Font.BOLD, ((int) fontSize) -
2);
            } catch (Exception e) {
               System.out.println("Error opening font file : " +
e.getMessage());
            }

            attCountNum.addAttribute(TextAttribute.FOREGROUND, Color.BLACK,
0, sCntNum.length());
```

```
            attCountNum.addAttribute(TextAttribute.BACKGROUND, Color.WHITE,
0, sCntNum.length());
            attCountNum.addAttribute(TextAttribute.FONT, fontCntNum, 0,
sCntNum.length());
            AttributedCharacterIterator aciCntNum =
attCountNum.getIterator();
            lbmCntNum = new LineBreakMeasurer(aciCntNum, fc);
         }

         //Background
         Font fontBg = null;
         try {
            fontBg = new Font(fontName, Font.PLAIN, ((int) fontSize) + 2);
         } catch (Exception e) {
            System.out.println("Error opening font file : " +
e.getMessage());
         }

         AttributedString attBk = new AttributedString(sBkground);
         attBk.addAttribute(TextAttribute.FOREGROUND, Color.WHITE, 0,
sBkground.length());
         attBk.addAttribute(TextAttribute.BACKGROUND, Color.WHITE, 0,
sBkground.length());
         attBk.addAttribute(TextAttribute.FONT, fontBg, 0,
sBkground.length());
         AttributedCharacterIterator aciBk = attBk.getIterator();
         LineBreakMeasurer lbmBk = new LineBreakMeasurer(aciBk, fc);

         while (lbm.getPosition() < line.length()) {
            BufferedImage lineBuffer = new BufferedImage(width, lineHeight,
BufferedImage.TYPE_INT_RGB);

            Graphics2D g0 = lineBuffer.createGraphics();
            g0.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
RenderingHints.VALUE_ANTIALIAS_ON);
            g0.setBackground(Color.WHITE);
            TextLayout layoutBk = lbmBk.nextLayout(wrappingWidth);
            int yBk = (int) layoutBk.getAscent();
            g0.setBackground(Color.WHITE);
            g0.setColor(Color.BLACK);
            layoutBk.draw(g0, 0, yBk);

            g1 = lineBuffer.createGraphics();
            g1.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
RenderingHints.VALUE_ANTIALIAS_ON);

            TextLayout layout = lbm.nextLayout(wrappingWidth);
            int y = (int) layout.getAscent();
            g1.setBackground(Color.WHITE);
            g1.setColor(Color.BLACK);
            layout.draw(g1, margin, y);

            if (line2.length() > 0)
            {
               Graphics2D g2 = lineBuffer.createGraphics();
               TextLayout layout2 = lbm2.nextLayout(wrappingWidth);
               int yy = (int) layout2.getAscent();
               g2.setBackground(Color.DARK_GRAY);
               g2.setColor(Color.DARK_GRAY);
               int stMargin = ((line2.length() * 300) / 100);
               stMargin = (400 - stMargin) + 10; //Start margin code at 415
and gap between histogram = 5 and start histogram at 10 if max.
               layout2.draw(g2, stMargin, yy);
            }
            if (countNum > 0)
            {
               //Line Counter
               Graphics2D g3 = lineBuffer.createGraphics();
```

```
                TextLayout layout3 = lbmCntNum.nextLayout(wrappingWidth);
                int yyy = (int) layout3.getAscent();
                g3.setBackground(Color.WHITE);
                int len = 0;
                if (line2.length() > 0)
                   len = line2.length();
                int stMargin = ((len * 300) / 100);
                stMargin = (400 - stMargin) + 10;
                int startMargin = (stMargin - sCntNum.length()) - 30;
                layout3.draw(g3, startMargin, yyy);
             }

             images.add(lineBuffer);
             lineCnt += lineHeight;
          }
       }

       try {
          saveImage(imgFilePath, fileCnt++);
       } catch (IOException e) {
          System.out.println("Error writing image : " + e.getMessage());
       }

       images.clear();
       lineCnt = 10;
    }

    static void saveImage(String fileName, int fileCnt) throws IOException {
       Graphics2D g;
       BufferedImage buffer = new BufferedImage(800, 800,
             BufferedImage.TYPE_INT_RGB);
       g = buffer.createGraphics();
       g.setBackground(Color.WHITE);
       g.setColor(Color.BLACK);
       g.clearRect(0, 0, 800, 800);

       for (int i = 0; i < images.size(); i++) {
          g.drawImage((BufferedImage) images.get(i), 0, 5//margin
                + (i * lineHeight), null);
       }

       buffer.flush();
       images.add(buffer);

       StringBuffer fullFileName = new StringBuffer(fileName);
       OutputStream out = new FileOutputStream(new File(fullFileName
             .toString()));
       ImageIO.write(buffer, "jpg", out);
       out.close();

       File imgFile = new File(fullFileName.toString());
       imgFile.setLastModified(ts.getTime());

       ts.setTime(ts.getTime() + (60 * 1000));
    }

    public ImageGenerator() {
    }

    public static void main(String[] args) {
    }
}
```

รหัสที่ ข.1 สร้างภาพฮิสโตแกรม