การปรับปรุงควิกซอร์ตด้วยตัวนำหน้าและตัวตามหลังของตัวหลัก

นาย ศิวัช เรืองพิภพ

ศูนย์วิทยทรัพยากร
จุฬาลงกรณ์มหาวิทยาลัย

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต

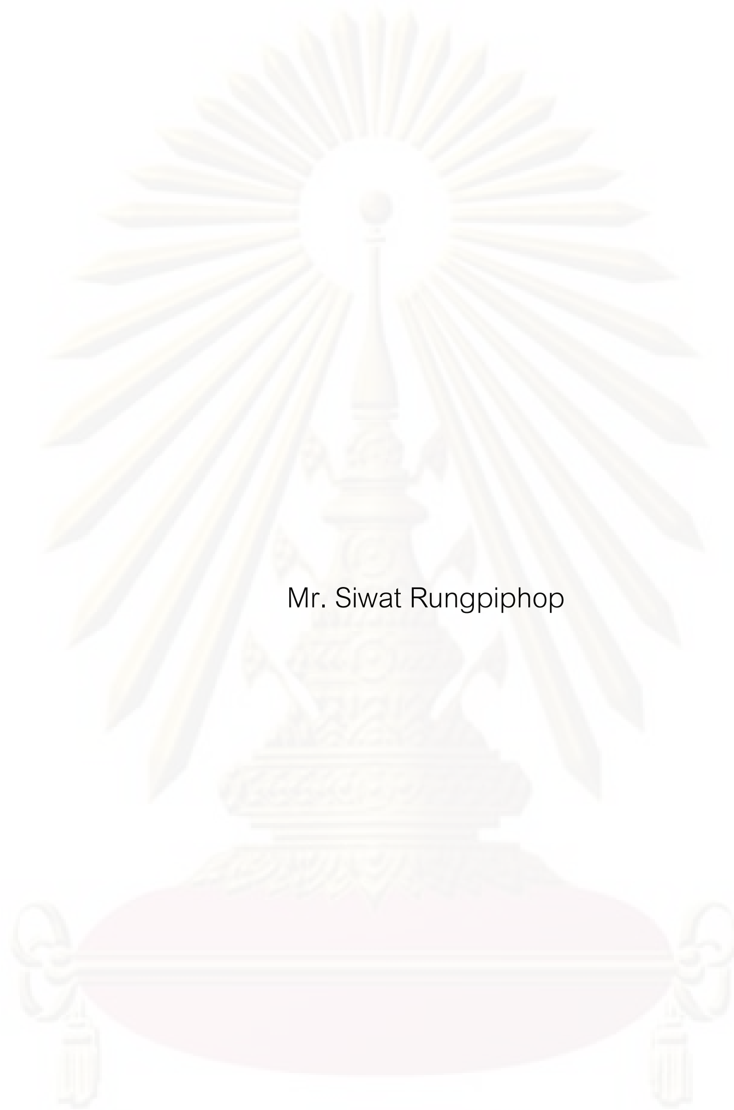
สาขาวิชาวิทยาการคณนา ภาควิชาคณิตศาสตร์

คณะวิทยาศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย

ปีการศึกษา 2552

ลิขสิทธิ์ของจุฬาลงกรณ์มหาวิทยาลัย

IMPROVEMENT OF QUICKSORT WITH PREDECESSOR AND SUCCESSOR OF PIVOT



Mr. Siwat Rungpiphop

ศูนย์วิทยทรัพยากร
จุฬาลงกรณ์มหาวิทยาลัย

A Thesis Submitted in Partial Fulfillment of the Requirements

for the Degree of Master of Science Program in Computational Science

Department of Mathematics

Faculty of Science

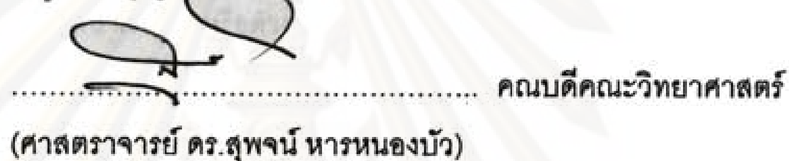
Chulalongkorn University

Academic Year 2009

Copyright of Chulalongkorn University

หัวข้อวิทยานิพนธ์	การปรับปรุงควิกซอร์ตด้วยตัวนำหน้าและตัวตามหลังของตัวหลัก
โดย	นายศิวัช เรืองพิภพ
สาขาวิชา	วิทยาการคณนา
อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก	ผู้ช่วยศาสตราจารย์ ดร.กรุง สินอภิรมย์สราญ

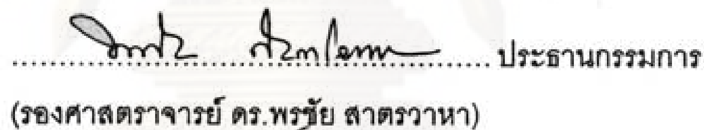
คณะวิทยาศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย อนุมัติให้นับวิทยานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญามหาบัณฑิต



.... คณบดีคณะวิทยาศาสตร์

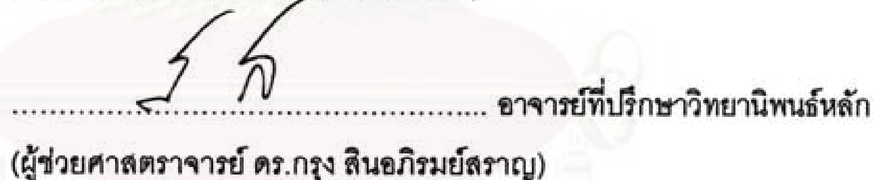
(ศาสตราจารย์ ดร.สุพจน์ หารหนองบัว)

คณะกรรมการสอบวิทยานิพนธ์



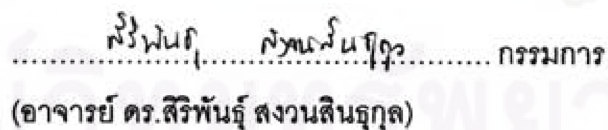
.... ประธานกรรมการ

(รองศาสตราจารย์ ดร.พรชัย สาตรวาหา)



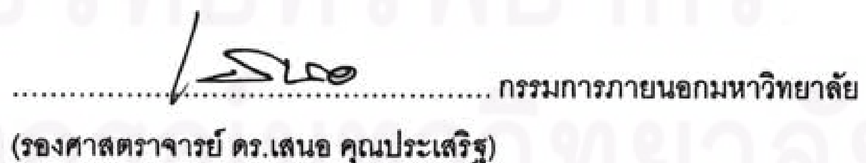
.... อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก

(ผู้ช่วยศาสตราจารย์ ดร.กรุง สินอภิรมย์สราญ)



.... กรรมการ

(อาจารย์ ดร.สิริพันธุ์ สงวนสินธุกุล)



.... กรรมการภายนอกมหาวิทยาลัย

(รองศาสตราจารย์ ดร.เสนอ คุณประเสริฐ)

ศูนย์วิทยทรัพยากร
จุฬาลงกรณ์มหาวิทยาลัย

ศิวัช เรืองพิภพ : การปรับปรุงควิกซอร์ตด้วยตัวนำหน้าและตัวตามหลังของตัวหลัก. (IMPROVEMENT OF QUICKSORT WITH PREDECESSOR AND SUCCESSOR OF PIVOT) อ. ที่ปรึกษาวิทยานิพนธ์หลัก : ผศ.ดร.กรุง สินอภิรมย์สราญ, 80 หน้า.

ควิกซอร์ตเป็นขั้นตอนวิธีการเรียงลำดับภายในที่นิยมใช้กันมากและมีงานวิจัยมากมายได้เสนอวิธีการปรับปรุงควิกซอร์ตให้มีประสิทธิภาพดีขึ้นเรื่อยมา งานวิจัยนี้เสนอควิกซอร์ตด้วยตัวประชิดตัวหลัก (APQsort) เป็นการปรับปรุงควิกซอร์ตให้มีประสิทธิภาพดีขึ้นสำหรับข้อมูลที่มีสมาชิกซ้ำกัน โดยการลดจำนวนครั้งที่เรียกฟังก์ชันเวียนเกิด APQsort ใช้ประโยชน์จากตัวนำหน้าตัวหลักหรือตัวตามหลังตัวหลักร่วมกับการแบ่งกั้นข้อมูลแบบสปลิตเอ็นสำหรับจัดการกับข้อมูลที่มีสมาชิกซ้ำกัน โดยเพิ่มการเก็บสมาชิกที่มีค่าเท่ากับตัวนำหน้าตัวหลักหรือตัวตามหลังตัวหลักแทนการเก็บเพียงสมาชิกที่มีค่าเท่ากับตัวหลัก และ APQsort สามารถตรวจสอบการเรียงลำดับของข้อมูลย่อยได้โดยพิจารณาจากตัวชี้ของตัวนำหน้าตัวหลักหรือตัวตามหลังตัวหลัก การทดสอบประสิทธิภาพของ APQsort ใช้ข้อมูลสี่แบบ ได้แก่ ข้อมูลแบบสุ่ม ข้อมูลที่เกือบเรียงลำดับ ข้อมูลที่เกือบเรียงลำดับแบบย้อนกลับ และข้อมูลแบบสุ่มที่มีสมาชิกซ้ำกัน โดยนำมาเปรียบเทียบกับ Mergesort Heapsort Quicksort (ควิกซอร์ตดั้งเดิม) qsort ซึ่งเป็นควิกซอร์ตที่ใช้การแบ่งกั้นข้อมูลแบบสปลิตเอ็น และ SWQuicksort ซึ่งเป็นขั้นตอนวิธีที่ปรับปรุงมาจากควิกซอร์ต จากการทดลองพบว่า APQsort ประมวลผลได้เร็วกว่า Mergesort Heapsort Quicksort qsort และ SWQuicksort สำหรับข้อมูลแบบสุ่มที่มีสมาชิกซ้ำกันเป็นจำนวนมาก และข้อมูลเรียงลำดับหรือข้อมูลเรียงลำดับแบบย้อนกลับ

ภาควิชา..............คณิตศาสตร์................ ลายมือชื่อนิสิต......................................

สาขาวิชา...........วิทยาการคณนา........... ลายมือชื่อ อ.ที่ปรึกษาวิทยานิพนธ์หลัก......................

ปีการศึกษา.........2552..........................

5072491723 : MAJOR COMPUTATIONAL SCIENCE
KEYWORDS : SORTING / QUICKSORT / PREDECESSOR PIVOT / SUCCESSOR PIVOT / SPLIT-END PARTITION

SIWAT RUNGPIPHOP : IMPROVEMENT OF QUICKSORT WITH PREDECESSOR AND SUCCESSOR OF PIVOT. THESIS ADVISOR : ASST. PROF. KRUNG SINAPIROMSARAN, 80 pp.

Quicksort is one of the most popular internal sorting algorithms. Many researches suggested various improvements of Quicksort. In this research, we propose Adjacent Pivot Quicksort (APQsort) which improves the performance of Quicksort for data with repeated elements by reducing the number of recursive calls. APQsort utilizes additional elements called "*pivot predecessor*" or "*pivot successor*" combined with the Split-end partition to handles the repeated elements keeping the elements which are equal to predecessor pivot or successor pivot instead of keeping only the elements which are equal to the pivot. APQsort can check the sorted sublist by consider pointer of predecessor pivot or successor pivot. We compare the performance of APQsort with the performance of Mergesort, Heapsort, the original Quicksort, qsort which is the Quicksort with split-end partition and SWQuicksort which is improvement of Quicksort in four different data sets; completely random data, nearly sorted data, nearly reverse sorted data and repeated element data. Our experiments show that APQsort significantly exhibits the faster running time for random data with a lot of repeat elements, sorted data and reverse sorted data than Mergesort, Heapsort, Quicksort, qsort and SWQuicksort.

Department : Mathematics Student's Signature ....
Field of Study : Computational Science Advisor's Signature ....
Academic Year : 2009

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้สามารถสำเร็จลุล่วงเป็นอย่างดีด้วยความกรุณาจากผู้ช่วยศาสตราจารย์ กรุง สินอภิรมย์สราญ ซึ่งให้การดูแลและคำปรึกษาที่เป็นประโยชน์ตลอดการดำเนินการวิจัย และขอขอบคุณกรรมการสอบวิทยานิพนธ์ทุกท่านที่ได้แนะนำสิ่งที่เป็นประโยชน์ต่องานวิจัยรวมถึงคณาจารย์ทุกท่านที่ได้ประสิทธิ์ประสาทวิชาความรู้แก่ผู้วิจัย

นอกจากนี้ผู้วิจัยขอขอบใจเพื่อนๆ สาขาวิทยาการคณนาทุกคนที่ให้ความช่วยเหลือและให้คำแนะนำตลอดมา

สุดท้ายขอขอบคุณคุณพ่อและคุณแม่ที่ให้การสนับสนุน ดูแลเอาใจใส่ข้าพเจ้าตลอดมาจนทำให้งานวิจัยสำเร็จลุล่วงไปด้วยดี

ศูนย์วิทยทรัพยากร
จุฬาลงกรณ์มหาวิทยาลัย

สารบัญ

หน้า

บทคัดย่อภาษาไทย		ง
บทคัดย่อภาษาอังกฤษ		จ
กิตติกรรมประกาศ		ฉ
สารบัญ		ช
สารบัญตาราง		ฌ
สารบัญภาพ		ญ

บทที่

1	บทนำ	1
	1.1 ความเป็นมาและความสำคัญของปัญหา	1
	1.2 วัตถุประสงค์ของงานวิจัย	4
	1.3 ขอบเขตของงานวิจัย	5
	1.4 ประโยชน์ของงานวิจัย	5
2	ทฤษฎีและงานวิจัยที่เกี่ยวข้อง	6
	2.1 ทฤษฎีที่เกี่ยวข้อง	6
	2.1.1 การเรียงลำดับ	6
	2.1.2 ควิกซอร์ต	14
	2.1.3 การวิเคราะห์ขั้นตอนวิธี	18
	2.1.4 ทฤษฎีบท Master	19
	2.1.5 การวิเคราะห์เวลาการทำงานของควิกซอร์ต	19
	2.1.6 มัธยฐานของสามตัว	23
	2.1.7 การแบ่งกั้นข้อมูลแบบสปลิตเอ็น	24
	2.2 งานวิจัยที่เกี่ยวข้อง	28
3	การปรับปรุงควิกซอร์ตด้วยตัวนำหน้าและตัวตามหลังของตัวหลัก	31
	3.1 ขั้นตอนวิธีควิกซอร์ตด้วยตัวนำหน้าและตัวตามหลังของตัวหลัก	31

ศูนย์วิทยทรัพยากร จุฬาลงกรณ์มหาวิทยาลัย

บทที่ หน้า

3.1.1 การแบ่งกั้นข้อมูลด้วยตัวนำหน้าและตัวตามหลังของตัวหลักที่ใช้การค้นหาโดยตรง.. 33
3.1.2 การแบ่งกั้นข้อมูลด้วยตัวนำหน้าและตัวตามหลังของตัวหลักที่ใช้การเรียงลำดับแบบฟอง... 34
3.3 ขั้นตอนวิธีควิกซอร์ตด้วยตัวนำหน้าตัวหลักและตัวตามหลังตัวหลักร่วมกับการแบ่งกั้นข้อมูลแบบสปลิตเอ็น.. 36
3.4 ขั้นตอนวิธีควิกซอร์ตด้วยตัวประชิดตัวหลัก.. 39

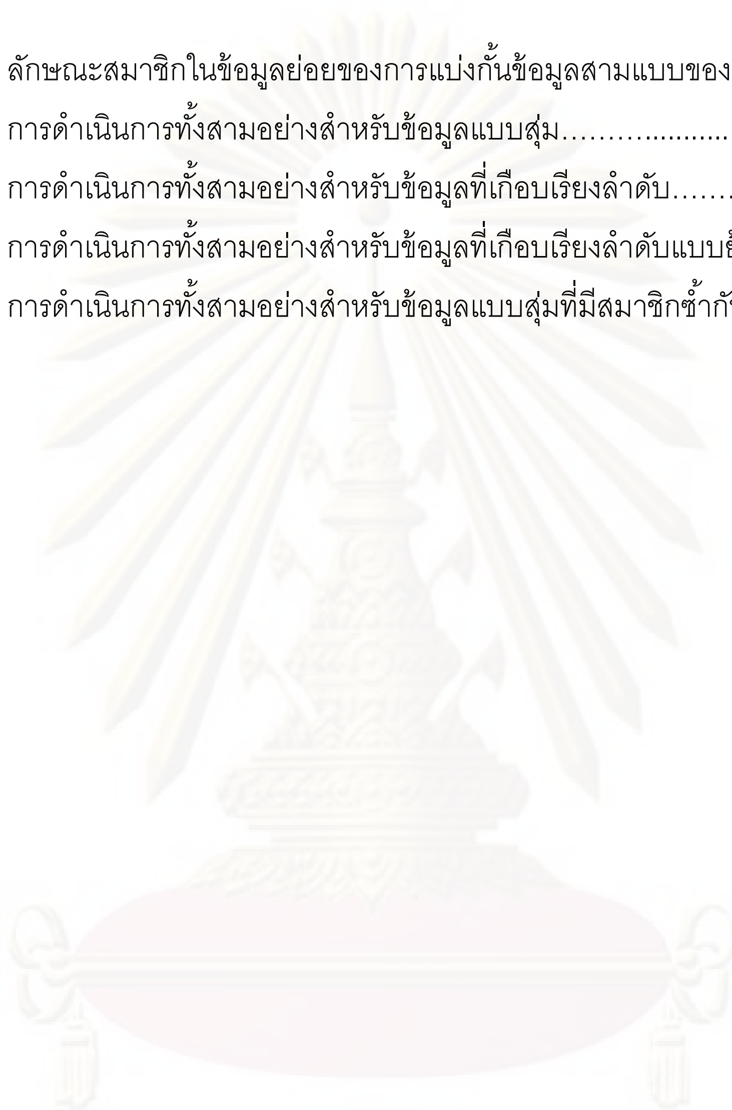
4 การทดลองและผลการทดลอง.. 48
4.1 การทดลอง.. 48
4.1.1 เครื่องมือและภาษาโปรแกรมที่ใช้ในการทดลอง........................ 48
4.1.2 ข้อมูลที่ใช้ทดลอง... 48
4.2 ผลการทดลอง... 49

5 สรุปผลการทดลอง.. 58

รายการอ้างอิง.. 60
ภาคผนวก.. 62
ภาคผนวก ก โปรแกรมการเรียงลำดับ.. 63
ภาคผนวก ข ผลการทดลอง…………………………………………………… 77
ประวัติผู้เขียนวิทยานิพนธ์.. 80

สารบัญตาราง

ตารางที่		หน้า
3.1	ลักษณะสมาชิกในข้อมูลย่อยของการแบ่งกั้นข้อมูลสามแบบของ APQsort.......	41
4.1	การดำเนินการทั้งสามอย่างสำหรับข้อมูลแบบสุ่ม………..............................	51
4.2	การดำเนินการทั้งสามอย่างสำหรับข้อมูลที่เกือบเรียงลำดับ……………………	53
4.3	การดำเนินการทั้งสามอย่างสำหรับข้อมูลที่เกือบเรียงลำดับแบบย้อนกลับ…….	55
4.4	การดำเนินการทั้งสามอย่างสำหรับข้อมูลแบบสุ่มที่มีสมาชิกซ้ำกัน……………	57



ศูนย์วิทยทรัพยากร
จุฬาลงกรณ์มหาวิทยาลัย

สารบัญภาพ

ภาพที่		หน้า
1.1	ประโยชน์ของการเรียงลำดับข้อมูล...	1
2.1	การทำงานของการเรียงลำดับแบบเลือก……………………………………....	7
2.2	การทำงานของการเรียงลำดับแบบแทรก…………………………………….....	8
2.3	การทำงานของการเรียงลำดับแบบฟอง…………………………………………	9
2.4	การทำงานของการเรียงลำดับแบบเชลล์………………………………………....	10
2.5	การทำงานของการเรียงลำดับแบบผสาน……………………………………….	11
2.6	การทำงานของควิกซอร์ต…………………………………………………………..	12
2.7	การทำงานของฮีปซอร์ต……………………………………………………………	13
2.8	การทำงานของการเรียงลำดับแบบเรดิกซ์………………………………………	13
2.9	การแบ่งกั้นข้อมูลของควิกซอร์ต…………………………………………………	14
2.10	การทำงานของควิกซอร์ตในกรณีดีที่สุด………………………………………..	20
2.11	การทำงานของควิกซอร์ตในกรณีแย่ที่สุดเมื่อตัวหลักเป็นค่าน้อยที่สุด…………	21
2.12	การแบ่งกั้นข้อมูลแบบสปลิตเอ็น………………………………………………..	24
2.13	ข้อมูลย่อยสามกลุ่มของการแบ่งกั้นข้อมูลแบบสปลิตเอ็น…….………………..	25
3.1	ลักษณะข้อมูลย่อยหลังการแบ่งกั้นข้อมูลแบบ PSQuicksort…………………..	32
3.2	การแบ่งกั้นข้อมูลด้วยตัวนำหน้าและตัวตามหลังของตัวหลักที่ใช้การค้นหาโดยตรง………………………………………………………………………..	33
3.3	การแบ่งกั้นข้อมูลด้วยตัวนำหน้าและตัวตามหลังของตัวหลักที่ใช้การเรียงลำดับแบบฟอง………………………………………………………………………	35
3.4	การแบ่งกั้นข้อมูลของ PSQsort………………………………………………..	36
3.5	ลักษณะข้อมูลย่อยหลังการแบ่งกั้นข้อมูลแบบ PSQsort………………………	37
3.6	ลักษณะของข้อมูลหลังจบการแบ่งกั้นข้อมูลด้วยตัวนำหน้าตัวหลัก……………	40
3.7	ลักษณะของข้อมูลหลังจบการแบ่งกั้นข้อมูลด้วยตัวตามหลังตัวหลัก…………..	40
3.8	การทำงานของควิกซอร์ตด้วยตัวประชิดตัวหลักในกรณีแย่ที่สุด………………..	47
4.1	การเปรียบเทียบเวลาการทำงานของข้อมูลแบบสุ่ม……………………………..	51
4.2	การเปรียบเทียบเวลาการทำงานของข้อมูลที่เกือบเรียงลำดับ…………………..	52
4.3	การเปรียบเทียบเวลาการทำงานของข้อมูลเกือบเรียงลำดับแบบย้อนกลับ……..	54

4.4 การเปรียบเทียบเวลาการทำงานของข้อมูลสุ่มที่มีสมาชิกซ้ำกัน……………….. 56

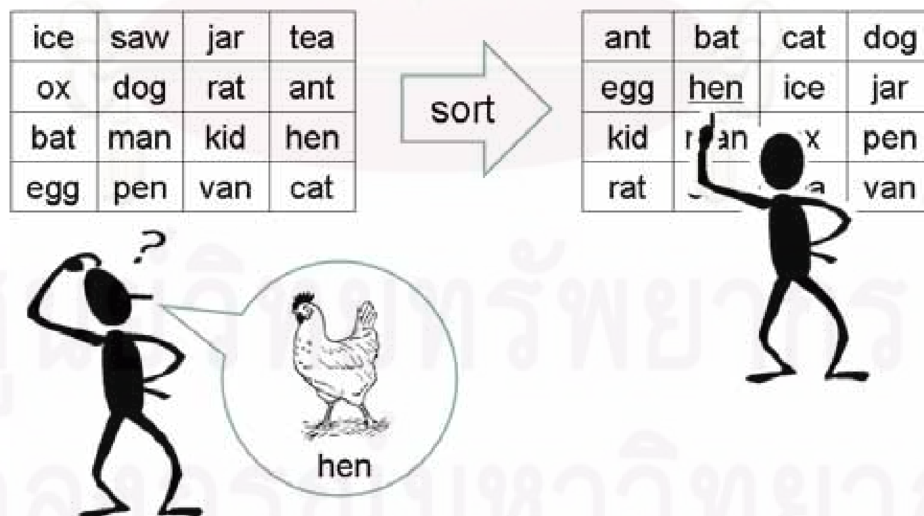
ศูนย์วิทยทรัพยากร
จุฬาลงกรณ์มหาวิทยาลัย

บทที่ 1

บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

การเรียงลำดับ (Sorting) เป็นกระบวนการประมวลผลเพื่อสลับลำดับของสมาชิกให้เรียงลำดับจากน้อยไปหามากหรือจากมากไปหาน้อย ประโยชน์ของการเรียงลำดับข้อมูลนอกจากจะเป็นการจัดเก็บข้อมูลให้เป็นระเบียบแล้วยังช่วยให้ผู้ใช้ค้นหาข้อมูลได้สะดวกรวดเร็วยิ่งขึ้น ยกตัวอย่างเช่น การจัดเรียงหมวดหมู่ของหนังสือในห้องสมุดต้องมีการจัดการกับรายละเอียดของหนังสือเรียงลำดับตามตัวอักษรของชื่อ เป็นต้น นอกจากนี้การเรียงลำดับยังถูกนำไปใช้เป็นขั้นตอนเบื้องต้นของปัญหาอื่นๆ อีกด้วย เช่น การค้นหาแบบทวิภาค (Binary search) [1] ปัญหาการเลือกกิจกรรม (Activity selection problem) [1] ปัญหาถุงเป้ที่คำตอบเป็นเศษส่วน (Fractional knapsack problem) [1] และการหาต้นไม้ทอดข้ามที่ต่ำที่สุด (Minimum spanning tree) [1] เป็นต้น



รูปที่ 1.1 ประโยชน์ของการเรียงลำดับข้อมูล

ศูนย์วิทยทรัพยากร
จุฬาลงกรณ์มหาวิทยาลัย

สำหรับคอมพิวเตอร์การเรียงลำดับเป็นการดำเนินการพื้นฐานที่ถูกเรียกใช้อยู่บ่อยครั้งซึ่งการเรียงลำดับนั้นจะช่วยจัดการให้ข้อมูลเป็นระเบียบและทำให้การเข้าถึงข้อมูลตามคีย์เร็วขึ้น โดยการเรียงลำดับทางคอมพิวเตอร์สามารถแยกออกเป็นสองประเภทตามการเรียกใช้หน่วยความจำหลักและหน่วยความจำสำรอง ได้แก่ การเรียงลำดับภายในและการเรียงลำดับภายนอก

การเรียงลำดับภายใน (Internal sorting) เป็นการเรียงลำดับข้อมูลที่ใช้เฉพาะหน่วยความจำหลัก ซึ่งข้อมูลทั้งหมดที่จะทำการเรียงลำดับนั้นต้องบันทึกลงบนหน่วยความจำหลักทั้งหมดก่อนการจัดเรียง และยังคงเหลือหน่วยความจำหลักบางส่วนไว้เพื่อดำเนินการเรียงลำดับข้อมูล

การเรียงลำดับภายนอก (External sorting) เป็นการเรียงลำดับข้อมูลที่ใช้หน่วยความจำหลักร่วมกับหน่วยความจำสำรอง ซึ่งข้อมูลทั้งหมดที่ทำการเรียงลำดับนั้นไม่จำเป็นต้องบันทึกลงบนหน่วยความจำหลักได้ทั้งหมด และต้องอาศัยสื่อบันทึกข้อมูลอื่น เช่น ดิสก์หรือเทปมาช่วยในการทำงาน

ในงานวิจัยนี้สนใจเฉพาะการเรียงลำดับภายในเนื่องจากในปัจจุบันเทคโนโลยีทางด้านฮาร์ดแวร์มีการพัฒนาอย่างต่อเนื่องทำให้เครื่องคอมพิวเตอร์ส่วนใหญ่มีหน่วยความจำหลักที่มีขนาดใหญ่และราคาถูกดังนั้นคอมพิวเตอร์จึงสามารถประมวลผลกับข้อมูลปริมาณมากได้ นอกจากนี้เวลาที่ใช้ในการประมวลผลของขั้นตอนวิธีการเรียงลำดับภายในจะใช้เวลาน้อยกว่าขั้นตอนวิธีการเรียงลำดับภายนอกเป็นอย่างมาก เนื่องจากเวลาที่ใช้ในการเข้าถึงข้อมูลในหน่วยความจำหลักซึ่งเป็นแบบอิเล็กโทรนิคเร็วกว่าการเข้าถึงข้อมูลในหน่วยความจำสำรองซึ่งเป็นแบบหมุน

ปัจจุบันขั้นตอนวิธีการเรียงลำดับภายในนั้นมีมากมายหลายวิธีซึ่งแต่ละวิธีก็มีข้อดีข้อเสียต่างกันออกไป งานวิจัยนี้เลือกที่จะปรับปรุงควิกซอร์ต (Quicksort) เนื่องจากควิกซอร์ตเป็นขั้นตอนวิธีการเรียงลำดับที่นิยมใช้กันทั่วไปและมีประสิทธิภาพเมื่อนำมาใช้กับข้อมูลจริง [1]

ควิกซอร์ตเป็นขั้นตอนวิธีการเรียงลำดับภายในที่อาศัยหลักการทำงานแบบแบ่งแยกและเอาชนะ (Divide and conquer) ควิกซอร์ตจะแบ่งข้อมูลออกเป็นสองข้อมูลย่อย โดยอาศัยสมาชิกตัวหนึ่งจากข้อมูลเป็นตัวแบ่งกั้น เราเรียกสมาชิกตัวนั้นว่า ตัวหลัก (Pivot) จากนั้นนำสมาชิกตัวอื่นมาเปรียบเทียบกับตัวหลัก โดยย้ายสมาชิกที่มีค่าน้อยกว่าตัวหลักไปทางด้านซ้ายและย้ายสมาชิก

ที่มีค่ามากกว่าตัวหลักไปทางด้านขวาซึ่งวิธีการนี้เรียกว่า การแบ่งกั้นข้อมูล (Partition) หลังจบการแบ่งกั้นข้อมูลจะได้ตำแหน่งที่ถูกต้องของตัวหลักกับข้อมูลย่อยทางซ้ายของตัวหลักซึ่งสมาชิกมีค่าน้อยกว่าตัวหลัก และข้อมูลย่อยทางขวาของตัวหลักซึ่งสมาชิกมีค่ามากกว่าตัวหลัก หลังจากนั้นทำการแบ่งกั้นข้อมูลซ้ำกับแต่ละข้อมูลย่อยจนกระทั่งไม่สามารถแบ่งข้อมูลได้อีก ควิกซอร์ตก็หยุด ผลลัพธ์ที่ได้ คือ ข้อมูลที่เรียงลำดับจากน้อยไปหามาก

ควิกซอร์ตเป็นวิธีการเรียงลำดับที่มีประสิทธิภาพมากสำหรับข้อมูลโดยทั่วไป และมีเวลาในการทำงานเป็น $\mathrm{O}(n\log n)$ ในกรณีดีที่สุดและกรณีเฉลี่ย แต่ควิกซอร์ตยังมีข้อด้อยที่สำคัญ คือ ในกรณีแย่ที่สุดควิกซอร์ตมีเวลาการทำงานเป็น $\mathrm{O}(n^2)$ ถ้าตัวหลักที่เลือกมีค่ามากหรือน้อยที่สุดในทุกรอบของการแบ่งกั้นข้อมูล จากปัญหาดังกล่าวและความต้องการเพิ่มประสิทธิภาพของควิกซอร์ตทำให้เกิดแนวคิดในการปรับปรุงควิกซอร์ตดังนี้

1) การปรับปรุงวิธีการเลือกตัวหลัก เป็นการเสนอเทคนิคในการเลือกสมาชิกที่ใช้สำหรับแบ่งกั้นข้อมูล โดยพยายามที่จะเลือกตัวหลักให้มีค่าใกล้เคียงกับค่ามัธยฐานของข้อมูล เพื่อเพิ่มประสิทธิภาพในการประมวลผลของควิกซอร์ต เช่น การเลือกตัวหลักแบบสุ่ม [2] การเลือกตัวหลักจากสมาชิกตรงกลาง [3] การใช้ค่ามัธยฐานของสมาชิกสามตัว [4] เป็นต้น
2) การปรับปรุงโดยใช้วิธีการเรียงลำดับแบบอื่นแทนควิกซอร์ตเมื่อข้อมูลมีขนาดเล็ก เป็นการศึกษาวิธีการเรียงลำดับอื่นที่สามารถประมวลได้เร็วกว่าควิกซอร์ตสำหรับข้อมูลที่มีขนาดเล็กเพื่อเพิ่มประสิทธิภาพในการประมวลผล เช่น การใช้การเรียงลำดับแบบแทรกแทนควิกซอร์ตเมื่อข้อมูลย่อยมีจำนวนสมาชิกน้อยกว่าหรือเท่ากับสิบตัว [5] เป็นต้น
3) การปรับปรุงควิกซอร์ตให้เหมาะสมกับข้อมูลที่มีลักษณะเฉพาะ เป็นการนำเทคนิคอื่นมาใช้ร่วมกับควิกซอร์ตเพื่อเพิ่มประสิทธิภาพในการประมวลผลสำหรับข้อมูลที่มีลักษณะเฉพาะ เช่น การใช้ควิกซอร์ตผสมกับการเรียงลำดับแบบฟองเพื่อตรวจสอบการเรียงลำดับของข้อมูลซึ่งวิธีการนี้เหมาะสำหรับข้อมูลที่เกือบเรียงลำดับ [6] ควิกซอร์ตด้วยการแบ่งกั้นข้อมูลแบบสปลิตเอ็นเป็นการปรับปรุงควิกซอร์ตให้เหมาะสำหรับข้อมูลที่มีสมาชิกซ้ำกัน [7] เป็นต้น

ประสิทธิภาพในการทำงานของควิกซอร์ตขึ้นอยู่กับปัจจัย 3 อย่าง ได้แก่ 1) จำนวนครั้งที่ใช้การเปรียบเทียบ 2) จำนวนครั้งที่ใช้การสลับที่ 3) จำนวนครั้งที่เรียกฟังก์ชันเวียนเกิด ซึ่งการเปรียบเทียบและการสลับที่จะเกิดขึ้นหลังจากมีการเรียกฟังก์ชันเวียนเกิด ดังนั้นผู้วิจัยคาดว่าขั้นตอนวิธีที่มีจำนวนครั้งที่เรียกฟังก์ชันเวียนเกิดน้อยก็จะทำให้การดำเนินการอื่นลดลงตามไปด้วย งานวิจัยนี้นำเสนอควิกซอร์ตด้วยตัวประชิดตัวหลัก (APQsort) เป็นวิธีการปรับปรุงควิกซอร์ตให้มีประสิทธิภาพดีขึ้นสำหรับข้อมูลที่มีสมาชิกซ้ำกัน โดยเน้นการลดจำนวนครั้งที่เรียกฟังก์ชันเวียนเกิด ซึ่งใช้ประโยชน์จากสมาชิกที่อยู่ตำแหน่งก่อนหน้าตัวหลักเรียกว่า ตัวนำหน้าตัวหลัก (Predecessor pivot) และสมาชิกที่อยู่ตำแหน่งตามหลังตัวหลักเรียกว่า ตัวตามหลังตัวหลัก (Successor pivot) ร่วมกับการแบ่งกั้นข้อมูลแบบสปลิตเอ็น (Split-end partition) [7] ซึ่งเป็นวิธีการแบ่งกั้นข้อมูลที่จัดการกับข้อมูลที่มีสมาชิกซ้ำกัน

หลักการทำงานของ APQsort ใช้ตัวนำหน้าตัวหลักและตัวตามหลังตัวหลักร่วมกับการแบ่งกั้นข้อมูลแบบสปลิตเอ็น โดยปกติการแบ่งกั้นข้อมูลแบบสปลิตเอ็นเก็บเฉพาะสมาชิกที่มีค่าเท่ากับตัวหลักเท่านั้น แต่ APQsort เพิ่มการเก็บสมาชิกที่มีค่าเท่ากับตัวนำหน้าตัวหลักหรือตัวตามหลังตัวหลัก ซึ่งจะรู้ตำแหน่งที่ถูกต้องของสมาชิกเพิ่มขึ้นจากเดิมทำให้จำนวนครั้งที่เรียกฟังก์ชันเวียนเกิดลดลง

1.2 **วัตถุประสงค์ของงานวิจัย**

งานวิจัยนี้เสนอควิกซอร์ตด้วยตัวประชิดตัวหลักเป็นวิธีการปรับปรุงควิกซอร์ตที่เน้นการลดจำนวนครั้งที่เรียกฟังก์ชันเวียนเกิด โดยใช้ประโยชน์จากตัวนำหน้าตัวหลักและตัวตามหลังตัวหลักร่วมกับการแบ่งกั้นข้อมูลแบบสปลิตเอ็น เพื่อลดจำนวนครั้งที่เรียกฟังก์ชันเวียนเกิดและเร่งเวลาการประมวลผลสำหรับข้อมูลที่มีสมาชิกซ้ำกัน เมื่อเปรียบเทียบกับควิกซอร์ตดั้งเดิมและควิกซอร์ตด้วยการแบ่งกั้นข้อมูลแบบสปลิตเอ็นจากงานวิจัยของ Bentley และ Mcilroy [7] ศึกษาข้อดีและข้อเสียของควิกซอร์ตด้วยตัวประชิดตัวหลัก

1.3 ขอบเขตของงานวิจัย

1) ข้อมูลที่นำมาเรียงลำดับเป็นแบบจำนวนเท่านั้น
2) การเรียงลำดับข้อมูลพิจารณาเฉพาะการเรียงลำดับจากน้อยไปหามากเท่านั้น
3) รหัสคำสั่งที่ใช้ในการทดลองเขียนด้วยภาษาซีเท่านั้น
4) การเปรียบเทียบประสิทธิภาพของขั้นตอนวิธีวัดจากเวลาการทำงาน จำนวนครั้งที่ใช้สลับข้อมูล จำนวนครั้งที่ใช้เปรียบเทียบข้อมูล และจำนวนครั้งที่เรียกฟังก์ชันเวียนเกิด

1.4 ประโยชน์ของงานวิจัย

ควิกซอร์ตด้วยตัวประชิดตัวหลักใช้จำนวนครั้งที่ใช้ฟังก์ชันเวียนเกิดลดลงและประมวลผลเร็วขึ้นสำหรับข้อมูลที่มีสมาชิกซ้ำกัน เมื่อเปรียบเทียบกับควิกซอร์ตดั้งเดิมและควิกซอร์ตด้วยการแบ่งกั้นแบบสปลิตเอ็นจากงานวิจัยของ Bentley และ Mcilroy

ศูนย์วิทยทรัพยากร จุฬาลงกรณ์มหาวิทยาลัย

บทที่ 2

ทฤษฎีและงานวิจัยที่เกี่ยวข้อง

2.1 ทฤษฎีที่เกี่ยวข้อง

งานวิจัยเรื่องการปรับปรุงควิกซอร์ตด้วยตัวนำหน้าและตัวตามหลังของตัวหลักมีความรู้และทฤษฎีพื้นฐานที่ควรทราบดังนี้ การเรียงลำดับ ควิกซอร์ต การวิเคราะห์ขั้นตอน ทฤษฎีบท Master การวิเคราะห์เวลาการทำงานของควิกซอร์ต วิธีมัธยฐานของสามตัว และการแบ่งกั้นข้อมูลแบบสปลิตเอ็น

2.1.1 การเรียงลำดับ

การเรียงลำดับ (Sorting) เป็นกระบวนการประมวลผลเพื่อสลับลำดับของสมาชิกให้เรียงลำดับจากน้อยไปหามากหรือจากมากไปหาน้อย ประโยชน์ของการเรียงลำดับ คือ ข้อมูลที่เรียงลำดับแล้วผู้ใช้สามารถถูกสืบค้นข้อมูลได้อย่างมีประสิทธิภาพโดยใช้การค้นหาแบบทวิภาค นอกจากนี้การเรียงลำดับยังถูกนำไปใช้เป็นขั้นตอนเบื้องต้นของปัญหาอื่นๆ อีกด้วย เช่น การค้นหาแบบทวิภาค (Binary search) [1] ปัญหาการเลือกกิจกรรม (Activity selection problem) [1] ปัญหาถุงเป้ที่คำตอบเป็นเศษส่วน (Fractional knapsack problem) [1] และการหาต้นไม้ทอดข้ามที่ต่ำที่สุด (Minimum spanning tree) [1] เป็นต้น

ในปัจจุบันขั้นตอนวิธีการเรียงลำดับมีการคิดค้นขึ้นมาหลายวิธี ซึ่งการเรียงลำดับทางคอมพิวเตอร์สามารถแบ่งเป็นสองประเภท ได้แก่ การเรียงลำดับภายในกับการเรียงลำดับภายนอก

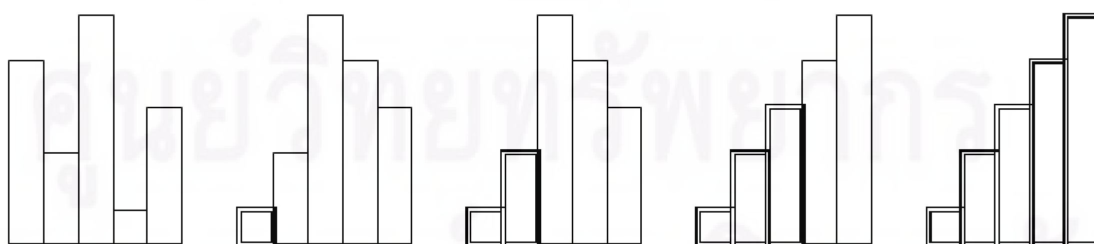
การเรียงลำดับภายใน (Internal sorting) [1, 8] เป็นการเรียงลำดับข้อมูลที่ใช้เฉพาะหน่วยความจำหลัก ซึ่งข้อมูลทั้งหมดที่จะทำการเรียงลำดับนั้นสามารถอ่านและบันทึกลงบนหน่วยความจำหลักได้ ตัวอย่างของขั้นตอนวิธีสำหรับการเรียงลำดับภายในซึ่งเป็นที่รู้จักโดยทั่วไป เช่น การเรียงลำดับแบบฟอง การเรียงลำดับเลือก การเรียงลำดับแบบแทรก การเรียงลำดับแบบเชลล์ การเรียงลำดับแบบผสาน ควิกซอร์ต ฮีปซอร์ต และการเรียงลำดับแบบเรดิกซ์ เป็นต้น

การเรียงลำดับภายนอก (External sorting) [1, 8] เป็นการเรียงลำดับข้อมูลที่ใช้หน่วยความจำหลักร่วมกับหน่วยความจำสำรอง ซึ่งข้อมูลทั้งหมดที่จะทำการเรียงลำดับนั้นไม่สามารถอ่านและบันทึกลงบนหน่วยความจำหลักได้ทั้งหมดจึงต้องอาศัยสื่อบันทึกข้อมูลอื่น เช่น ดิสก์ หรือ เทป มาช่วยในการทำงาน โดยการเรียงลำดับภายนอกจะต้องแบ่งข้อมูลออกเป็นส่วนย่อย จากนั้นข้อมูลย่อยแต่ละส่วนจะถูกเรียงลำดับด้วยการเรียงลำดับภายใน และบันทึกข้อมูลย่อยที่เรียงลำดับแล้วลงในหน่วยความจำสำรองเป็นการชั่วคราวเพื่อรอที่จะรวมกับข้อมูลส่วนอื่นที่เรียงลำดับเสร็จแล้ว ตัวอย่างของขั้นตอนวิธีสำหรับการเรียงลำดับภายนอกซึ่งเป็นที่รู้จักโดยทั่วไป เช่น การเรียงลำดับภายนอกแบบผสาน เป็นต้น

จากที่ได้กล่าวไว้ในบทแรก งานวิจัยนี้สนใจเฉพาะการเรียงลำดับภายใน ขั้นตอนวิธีสำหรับการเรียงลำดับภายในที่ยังปรากฏอยู่ในปัจจุบันมีดังนี้

1) การเรียงลำดับแบบเลือก (Selection sort)

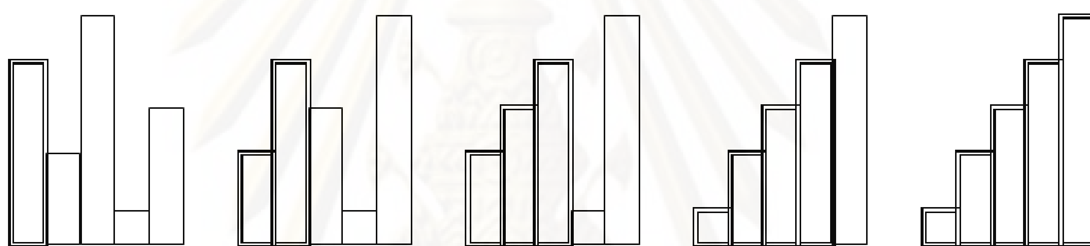
การเรียงลำดับแบบเลือก [8] เป็นวิธีการเรียงลำดับซึ่งทำการย้ายสมาชิกมาวางในตำแหน่งที่ถูกต้องรอบละหนึ่งตัว โดยทำซ้ำขั้นตอนเดิมไปจนกระทั่งย้ายสมาชิกมาเก็บในตำแหน่งที่ถูกต้องครบทุกตัว สำหรับการเรียงลำดับจากน้อยไปมากมีขั้นตอนดังนี้ ในรอบแรกทำการค้นหาสมาชิกที่มีค่าน้อยที่สุดแล้วย้ายมาเก็บไว้ที่ตำแหน่งแรก รอบต่อไปทำการค้นหาสมาชิกที่มีค่าน้อยรองลงมาแล้วย้ายมาเก็บไว้ที่ตำแหน่งที่สอง ทำซ้ำเช่นนี้ไปจนกระทั่งย้ายสมาชิกมาเก็บในตำแหน่งที่ถูกต้องครบทุกตัว เวลาการทำงานของการเรียงลำดับแบบเลือกเป็น $O(n^2)$ สำหรับทุกกรณี



รูปที่ 2.1 การทำงานของการเรียงลำดับแบบเลือก

2) การเรียงลำดับแบบแทรก (Insertion sort)

การเรียงลำดับแบบแทรก [8] เป็นวิธีการเรียงลำดับซึ่งทำการแทรกสมาชิกเข้าไปในข้อมูลที่เรียงลำดับแล้วยังคงการเรียงลำดับของข้อมูลนั้นอยู่ จากนั้นทำซ้ำขั้นตอนเดิมไปจนกระทั่งแทรกสมาชิกไปอยู่ในตำแหน่งที่ถูกต้องครบทุกตัว ซึ่งวิธีการนี้จะคล้ายกับการจัดเรียงไพ่ในมือ นั่นคือในขณะเล่นไพ่เรามักจะเรียงไพ่ในมือตามลำดับตัวเลข และเมื่อหยิบไพ่ใบใหม่เข้ามาเพิ่มจะแทรกลงไประหว่างไพ่ซึ่งเรียงลำดับไว้แล้ว เวลาการทำงานเฉลี่ยของการเรียงลำดับแบบแทรกเป็น $O(n^2)$ และเวลาการทำงานสำหรับกรณีที่ดีที่สุดเป็น $O(n)$ ถ้าข้อมูลเข้ามีการเรียงลำดับที่ถูกต้องแล้ว

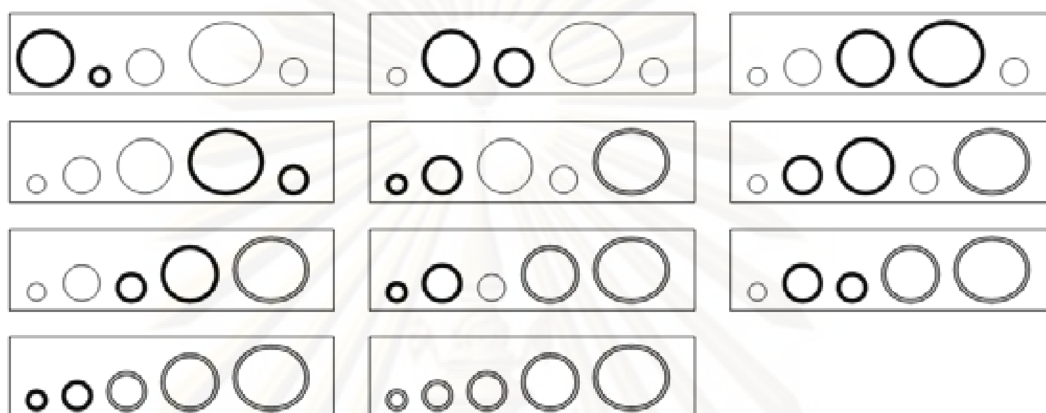


รูปที่ 2.2 การทำงานของการเรียงลำดับแบบแทรก

3) การเรียงลำดับแบบฟอง (Bubble sort)

การเรียงลำดับแบบฟอง [8, 9] เป็นวิธีการเรียงลำดับซึ่งทำการดันสมาชิกให้ไปอยู่ในตำแหน่งที่ถูกต้องรอบละหนึ่งตัว โดยจะเปรียบเทียบสมาชิกในตำแหน่งที่อยู่ติดกัน ถ้าสมาชิกทั้งสองไม่อยู่ในลำดับที่ถูกต้องให้สลับที่กัน ทำซ้ำขั้นตอนเดิมไปจนกระทั่งดันสมาชิกมาอยู่ในตำแหน่งที่ถูกต้องครบทุกตัว สำหรับการเรียงลำดับจากน้อยไปมากมีขั้นตอนดังนี้ ในรอบแรกเริ่มจากเปรียบเทียบสมาชิกที่ตำแหน่งแรกกับตำแหน่งที่สอง ถ้าข้อมูลไม่เรียงลำดับก็จะสลับที่กัน จากนั้นทำซ้ำขั้นตอนเดิมกับสมาชิกในตำแหน่งถัดไปจนถึงสมาชิกตัวสุดท้ายจะได้ค่าที่ตำแหน่งสุดท้ายเป็นค่าที่มากที่สุด จากนั้นในรอบต่อมาทำซ้ำเหมือนกับรอบแรกจนถึงสมาชิกตัวรองสุดท้ายก็จะได้ค่าที่ตำแหน่งรองสุดท้ายเป็นค่าที่มากรองลงมา ทำซ้ำกับขั้นตอนเดิมจนกระทั่งดัน

สมาชิกแต่ละตัวไปอยู่ตำแหน่งที่ถูกต้องจนครบ เวลาการทำงานของการเรียงลำดับแบบฟองเป็น $O(n^2)$ สำหรับทุกกรณี

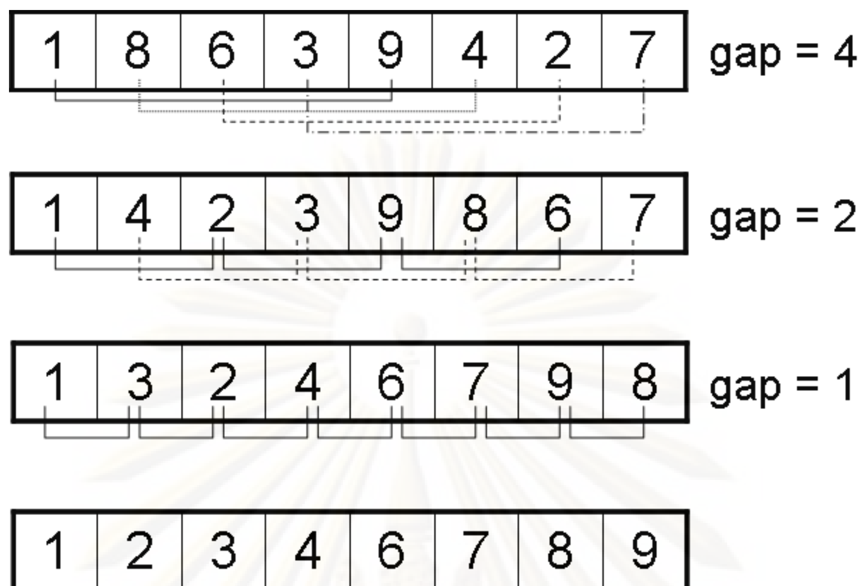


รูปที่ 2.3 การทำงานของการเรียงลำดับแบบฟอง

4) การเรียงลำดับแบบเชลล์ (Shell sort)

การเรียงลำดับแบบเชลล์ [8, 10] ถูกเสนอขึ้นโดย Donald Shell ในปี 1959 เป็นวิธีการเรียงลำดับที่ทำให้ข้อมูลเรียงลำดับมากขึ้นจนกระทั่งเรียงลำดับทั้งหมด โดยใช้การเรียงลำดับสมาชิกที่อยู่ในตำแหน่งห่างกัน (gap) ซึ่งระยะห่างนี้จะลดลงทีละครึ่งจนกระทั่งเท่ากับหนึ่ง โดยอาศัยการเรียงลำดับแบบแทรกช่วยในการเรียงลำดับ เวลาการทำงานของการเรียงลำดับแบบเชลล์เป็น $O(n^2)$ สำหรับกรณีแย่ที่สุด

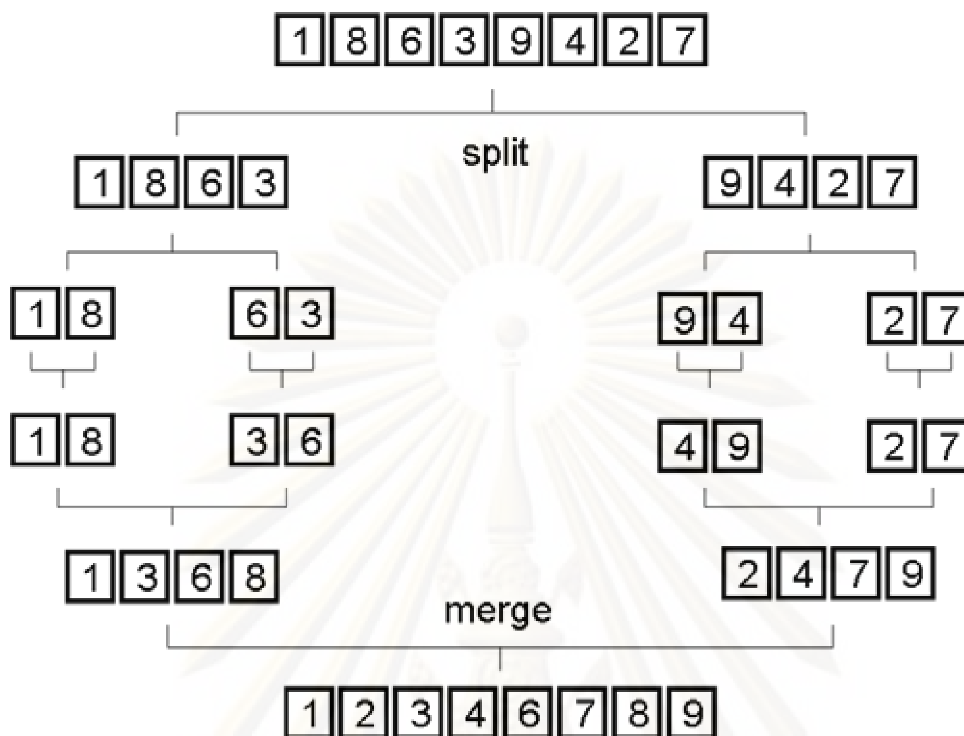
ศูนย์วิทยทรัพยากร
จุฬาลงกรณ์มหาวิทยาลัย



รูปที่ 2.4 การทำงานของการเรียงลำดับแบบเชลล์

5) การเรียงลำดับแบบผสาน (Merge sort)

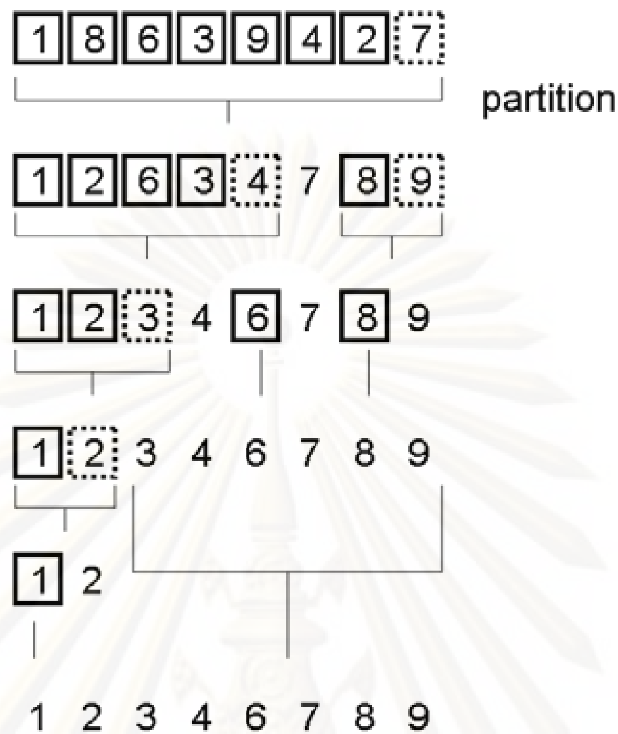
การเรียงลำดับแบบผสาน [8] ถูกเสนอขึ้นโดย John von Neumann ในปี 1945 เป็นวิธีการเรียงลำดับที่อาศัยหลักการทำงานแบบแบ่งแยกและเอาชนะ หลักการทำงานของการเรียงลำดับแบบผสานทำการแบ่งข้อมูลออกเป็นสองส่วนซึ่งมีจำนวนสมาชิกเท่ากัน โดยจะแบ่งข้อมูลไปจนกระทั่งข้อมูลย่อยมีสมาชิกน้อยกว่าหรือเท่ากับสองตัว จากนั้นนำข้อมูลย่อยแต่ละชุดมาเรียงลำดับ เมื่อได้ข้อมูลย่อยที่เรียงลำดับเรียบร้อยแล้วจึงนำข้อมูลย่อยแต่ละชุดมาผสานกันทีละสองชุด ซึ่งทำการผสานไปจนกระทั่งเหลือข้อมูลเพียงชุดเดียวจะได้ข้อมูลที่เรียงลำดับตามต้องการ เวลาการทำงานของการเรียงลำดับแบบผสานเป็น $\mathrm{O}(n\log n)$ สำหรับทุกกรณี



รูปที่ 2.5 การทำงานของการเรียงลำดับแบบผสาน

6) ควิกซอร์ต (Quicksort)

ควิกซอร์ต [1, 2, 8, 11] ถูกเสนอขึ้นโดย C.A.R. Hoare ในปี 1961 เป็นวิธีการเรียงลำดับที่อาศัยหลักการทำงานแบบแบ่งแยกและเอาชนะ เริ่มต้นกำหนดสมาชิกหนึ่งตัวเพื่อใช้สำหรับแบ่งกั้นข้อมูล เรียกสมาชิกตัวนี้ว่าตัวหลัก (Pivot) จากนั้นทำการเปรียบเทียบสมาชิกตัวอื่นกับตัวหลัก โดยย้ายสมาชิกที่มีค่าน้อยกว่าตัวหลักไปทางด้านซ้ายและย้ายสมาชิกที่มีค่ามากกว่าหรือเท่ากับตัวหลักไปทางด้านขวา เรียกขั้นตอนนี้ว่า การแบ่งกั้นข้อมูล (Partition) หลังจบการแบ่งกั้นข้อมูล ตัวหลักจะอยู่ ณ ตำแหน่งที่ถูกต้อง และได้ข้อมูลย่อยที่อยู่ทางซ้ายของตัวหลักซึ่งสมาชิกในกลุ่มมีค่าน้อยกว่าตัวหลัก กับข้อมูลย่อยที่อยู่ทางขวาของตัวหลักซึ่งสมาชิกในกลุ่มมีค่ามากกว่าหรือเท่ากับตัวหลัก ทำการแบ่งกั้นข้อมูลซ้ำกับแต่ละข้อมูลย่อยจนกระทั่งไม่สามารถแบ่งข้อมูลได้อีก ควิกซอร์ตก็จะหยุด ผลลัพธ์ที่ได้เป็นข้อมูลที่เรียงลำดับจากน้อยไปหามาก เวลาการทำงานเฉลี่ยของควิกซอร์ตเป็น $\mathrm{O}(n \log n)$ และเวลาการทำงานสำหรับกรณีแย่ที่สุดเป็น $\mathrm{O}(n^2)$ ถ้าตัวหลักมีค่ามากที่สุดหรือน้อยที่สุด



รูปที่ 2.6 การทำงานของควิกซอร์ต

7) ฮีปซอร์ต (Heapsort)

ฮีปซอร์ต [1, 12] ถูกเสนอขึ้นโดย J.W.J. Williams ในปี 1964 เป็นวิธีการเรียงลำดับที่คล้ายกับการเรียงลำดับแบบเลือก แต่ใช้โครงแบบฮีปมาช่วยในการเรียงลำดับข้อมูล โครงสร้างแบบฮีป คือ ต้นไม้แบบทวิภาคสมบูรณ์ซึ่งมีสมบัติ คือ สมาชิกที่ parent node มีค่ามากกว่าหรือเท่ากับสมาชิกที่ child node เสมอ สำหรับการเรียงลำดับจากน้อยไปมากมีขั้นตอนดังนี้ เริ่มต้นจากนำข้อมูลที่ต้องการเรียงลำดับมาสร้างฮีปซึ่งได้สมาชิกที่ root node เป็นสมาชิกที่มีค่ามากที่สุดและย้ายสมาชิกนั้นไปอยู่ในตำแหน่งที่ถูกต้อง จากนั้นนำข้อมูลที่เหลือมาทำให้เป็นฮีปซึ่งได้สมาชิกที่ root node เป็นสมาชิกที่มีค่ามากรองลงมาและย้ายสมาชิกนั้นไปอยู่ในตำแหน่งที่ถูกต้อง ทำซ้ำขั้นตอนเดิมไปจนกระทั่งย้ายสมาชิกไปอยู่ในตำแหน่งที่ถูกต้องครบทุกตัว เวลาการทำงานของฮีปซอร์ตเป็น $\mathrm{O}(n \log n)$ สำหรับทุกกรณี

รูปที่ 2.7 การทำงานของฮีปซอร์ต

8) การเรียงลำดับแบบเรดิกซ์ (Radix sort)

การเรียงลำดับแบบเรดิกซ์ [1] เป็นการเรียงลำดับที่พิจารณาข้อมูลทีละหลัก โดยเริ่มพิจารณาจากหลักที่มีค่าน้อยที่สุดก่อนโดยเรียงตามลำดับการเข้ามา จากนั้นทำซ้ำขั้นตอนข้างต้นโดยพิจารณาหลักที่มีค่าน้อยรองลงมา ทำไปจนครบทุกหลักจะได้ข้อมูลที่เรียงลำดับจากน้อยไปมาก เวลาการทำงานของการเรียงลำดับแบบเรดิกซ์เป็น $O(kn)$ ซึ่ง k เป็นจำนวนค่าที่แตกต่างของข้อมูล เช่น ข้อมูลตัวเลข k = 10 ข้อมูลภาษาอังกฤษ k = 26

	หลักหน่วย	หลักสิบ	หลักร้อย
153	153	313	153
313	313	844	245
245	844	245	299
299	894	153	313
844	245	894	698
894	698	698	844
698	299	299	894

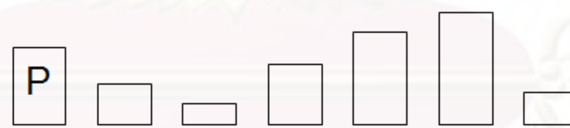
รูปที่ 2.8 การทำงานของการเรียงลำดับแบบเรดิกซ์

ศูนย์วิทยทรัพยากร
จุฬาลงกรณ์มหาวิทยาลัย

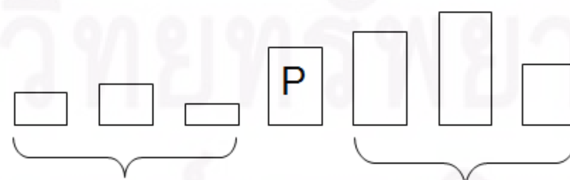
2.1.2 ควิกซอร์ต

ควิกซอร์ตถูกเสนอขึ้นโดย C.A.R. Hoare ในปี ค.ศ.1961 [2] เป็นขั้นตอนวิธีการเรียงลำดับข้อมูลภายในที่อาศัยการทำงานแบบแบ่งแยกและเอาชนะซึ่งอาศัยตัวหลัก ในการแบ่งข้อมูลออกเป็นสองกลุ่ม คือ กลุ่มที่สมาชิกมีค่าน้อยกว่าตัวหลักกับกลุ่มที่สมาชิกมีค่ามากกว่าตัวหลัก

หลักการทำงานของควิกซอร์ต เริ่มต้นจากกำหนดสมาชิกมาหนึ่งตัวเพื่อใช้สำหรับแบ่งกั้นข้อมูลเรียกสมาชิกตัวนี้ว่า ตัวหลัก (Pivot) จากนั้นทำการเปรียบเทียบสมาชิกตัวอื่นกับตัวหลักโดยย้ายสมาชิกที่มีค่าน้อยกว่าตัวหลักไปทางด้านซ้ายและย้ายสมาชิกที่มีค่ามากกว่าหรือเท่ากับตัวหลักไปทางด้านขวาซึ่งเรียกขั้นตอนนี้ว่า การแบ่งกั้นข้อมูล (Partition) หลังจบการแบ่งกั้นข้อมูลตัวหลักจะอยู่ ณ ตำแหน่งที่ถูกต้อง และกลุ่มข้อมูลย่อยที่อยู่ทางซ้ายของตัวหลักซึ่งสมาชิกในกลุ่มมีค่าน้อยกว่าตัวหลักกับกลุ่มข้อมูลย่อยที่อยู่ทางขวาของตัวหลักซึ่งสมาชิกในกลุ่มมีค่ามากกว่าหรือเท่ากับตัวหลัก ดังรูปที่ 2.9 จากนั้นทำซ้ำขั้นตอนข้างต้นกับแต่ละกลุ่มข้อมูลย่อยจนกระทั่งไม่สามารถแบ่งข้อมูลได้อีกควิกซอร์ตก็จะหยุด ผลลัพธ์ที่ได้จะเป็นข้อมูลที่เรียงลำดับจากน้อยไปหามาก



ซึ่ง P คือ ตัวหลัก



รูปที่ 2.9 การแบ่งกั้นข้อมูลของควิกซอร์ต

การทำงานของควิกซอร์ตมีขั้นตอนวิธีดังนี้

1) เริ่มต้นกำหนดสมาชิกหนึ่งตัวเป็นตัวหลัก และตัวชี้ i ซึ่งชี้จากทางด้านซ้ายของข้อมูล กับตัวชี้ j ซึ่งชี้จากทางด้านขวาของข้อมูล ในกรณีที่ตัวหลักเป็นตัวซ้ายสุด ตัวชี้ i จะเริ่มชี้ที่สมาชิกตำแหน่งที่สองจากซ้าย และในกรณีตัวหลักเป็นตัวขวาสุด ตัวชี้ j จะเริ่มชี้ที่สมาชิกตำแหน่งที่สองจากขวา

2) ทำการเปรียบเทียบสมาชิกที่ตัวชี้ i กับตัวหลัก ถ้าสมาชิกที่ตัวชี้ i มีค่าน้อยกว่าตัวหลักแล้วตัวชี้ i จะขยับไปทางขวาหนึ่งช่อง โดยทำการเปรียบเทียบซ้ำจนกระทั่งตรวจเจอสมาชิกที่มีค่ามากกว่าหรือเท่ากับตัวหลักจึงหยุด จากนั้นทำการเปรียบเทียบสมาชิกที่ตัวชี้ j กับตัวหลัก ถ้าสมาชิกที่ตัวชี้ j มีค่ามากกว่าหรือเท่ากับตัวหลักแล้วตัวชี้ j จะขยับไปทางด้านซ้ายหนึ่งช่อง โดยทำการเปรียบเทียบซ้ำจนกระทั่งตรวจเจอสมาชิกที่มีค่าน้อยกว่าตัวหลักจึงหยุด ถ้าตัวชี้ i มีค่าน้อยกว่าตัวชี้ j แล้วสลับที่สมาชิกของตัวชี้ i กับตัวชี้ j จากนั้นทำซ้ำขั้นตอนข้างต้นไปจนกระทั่งตัวชี้ i มีค่ามากกว่าหรือเท่ากับตัวชี้ j จึงหยุดและสลับที่สมาชิกของตัวชี้ i กับตัวหลัก ซึ่งขั้นตอนนี้เรียกว่า การแบ่งกั้นข้อมูล

3) หลังจบการแบ่งกั้นข้อมูลตัวหลักจะอยู่ ณ ตำแหน่งที่ถูกต้อง และข้อมูลย่อยทางด้านซ้ายของตัวหลักซึ่งสมาชิกในกลุ่มมีค่าน้อยกว่าตัวหลักกับข้อมูลย่อยทางด้านขวาของตัวหลักซึ่งสมาชิกในกลุ่มมีค่ามากกว่าหรือเท่ากับตัวหลัก ทำการแบ่งกั้นข้อมูลซ้ำกับแต่ละข้อมูลย่อยจนกระทั่งไม่สามารถแบ่งข้อมูลได้อีกควิกซอร์ตก็จะหยุด ผลลัพธ์ที่ได้เป็นข้อมูลที่เรียงลำดับจากน้อยไปหามาก

จากหลักการที่กล่าวมาข้างต้นสามารถเขียนเป็นรหัสคำสั่งได้ดังนี้ ในที่นี้กำหนดให้ตัวหลักเป็นสมาชิกตำแหน่งขวาของข้อมูลเสมอ

```
Function Quicksort (A[ ], left, right)
    IF (right > left) THEN
        pivot_index = Partition (A, left, right);
        Quicksort (A, left, pivot_index – 1);
        Quicksort (A, pivot_index + 1, right);
    ENDIF
End Quicksort

Function Partition (A[ ], left, right)
    i = left;  j = right – 1;
```

ศูนย์วิทยทรัพยากร
จุฬาลงกรณ์มหาวิทยาลัย

```
    pivot = A[right];

    LOOP
        WHILE (A[i] < pivot) DO
            i = i + 1;
        ENDWHILE
        WHILE (A[i] >= pivot) DO
            j = j – 1;
        ENDWHILE
        IF (i >= j) THEN
            Exit LOOP;
        ENDIF
        Swap A[i] and A[j];
    ENDLOOP

    Move pivot to correct position;
    Return pivot index;
End Partition
```

ตัวอย่างการทำงานของควิกซอร์ต โดยกำหนดข้อมูลดังนี้ 2, 8, 5, 1, 4, 7, 3, 9, 6

เริ่มต้นกำหนดให้ตัวหลักเป็นสมาชิกตัวท้ายของข้อมูลซึ่งมีค่าเท่ากับ 6 และกำหนดให้ i เป็นตัวชี้ที่ตำแหน่งแรกและ j เป็นตัวชี้ที่ตำแหน่งรองสุดท้าย

2	8	5	1	4	7	3	9	<u>6</u>
i							*j*	

ทำการเปรียบเทียบสมาชิกที่ตัวชี้ i กับตัวหลัก โดยขยับตัวชี้ i มาทางด้านขวาจนกระทั่งเจอสมาชิกที่มีค่ามากกว่าหรือเท่ากับ 6 โดยตัวชี้ i หยุดที่สมาชิกตำแหน่งที่สองซึ่งมีค่าเท่ากับ 8

2	8	5	1	4	7	3	9	<u>6</u>
	i						*j*	

ทำการเปรียบเทียบสมาชิกที่ตัวชี้ j กับตัวหลัก โดยขยับตัวชี้ j มาทางด้านซ้ายจนกระทั่งเจอสมาชิกที่มีค่าน้อย 6 โดยตัวชี้ j หยุดที่สมาชิกตำแหน่งที่เจ็ดซึ่งมีค่าเท่ากับ 3

2	8	5	1	4	7	3	9	<u>6</u>
	i					*j*		

ทำการเปรียบเทียบตัวชี้ i กับตัวชี้ j โดยตัวชี้ i ชี้ที่ตำแหน่งที่สองซึ่งน้อยกว่าตัวชี้ j ชี้ที่ตำแหน่งที่เจ็ด ดังนั้นทำการสลับที่สมาชิกระหว่างตัวชี้ i กับตัวชี้ j

2	3	5	1	4	7	8	9	6
	i					j		

ทำการเปรียบเทียบสมาชิกที่ตัวชี้ i กับตัวหลัก โดยขยับตัวชี้ i มาทางด้านขวาจนกระทั่งเจอสมาชิกที่มีค่ามากกว่าหรือเท่ากับ 6 โดยตัวชี้ i หยุดที่สมาชิกตำแหน่งที่หกซึ่งมีค่าเท่ากับ 7

2	3	5	1	4	7	8	9	6
					i	j		

ทำการเปรียบเทียบสมาชิกที่ตัวชี้ j กับตัวหลัก โดยขยับตัวชี้ j มาทางด้านซ้ายจนกระทั่งเจอสมาชิกที่มีค่าน้อย 6 โดยตัวชี้ j หยุดที่สมาชิกตำแหน่งที่ห้าซึ่งมีค่าเท่ากับ 4

2	3	5	1	4	7	8	9	6
				j	i			

ทำการเปรียบเทียบตัวชี้ i กับตัวชี้ j โดยตัวชี้ i ชี้ที่ตำแหน่งที่หกซึ่งมากกว่าตัวชี้ j ชี้ที่ตำแหน่งที่ห้า ดังนั้นทำการสลับที่สมาชิกระหว่างตัวชี้ i กับตัวหลัก

2	3	5	1	4	6	8	9	7

เมื่อถึงขั้นตอนนี้ตัวหลักจะอยู่ ณ ตำแหน่งที่ถูกต้อง และได้ข้อมูลย่อยทางซ้ายของตัวหลักซึ่งสมาชิกมีค่าน้อยกว่าตัวหลักกับข้อมูลย่อยทางขวาของตัวหลักซึ่งสมาชิกมีค่ามากกว่าหรือเท่ากับตัวหลัก จากนั้นทำการแบ่งข้อมูลซ้ำกับแต่ละข้อมูลย่อยจนกระทั่งไม่สามารถแบ่งข้อมูลได้อีกควิกซอร์ตก็จะหยุด ผลลัพธ์ที่ได้ คือ ข้อมูลที่เรียงลำดับจากน้อยไปมาก

ศูนย์วิทยทรัพยากร
จุฬาลงกรณ์มหาวิทยาลัย

2.1.3 การวิเคราะห์ขั้นตอนวิธี

การวิเคราะห์ขั้นตอนวิธี [1] เป็นการคาดการณ์ปริมาณทรัพยากรที่ใช้ของขั้นตอนวิธี เช่น หน่วยความจำหรือเวลาในการประมวลผล เป็นต้น โดยทั่วไปการวิเคราะห์การทำงานของขั้นตอนวิธีจะถูกนิยามเป็นฟังก์ชันตามขนาดข้อมูล ซึ่งจะพิจารณาการเติบโตของฟังก์ชันเมื่อข้อมูลมีขนาดใหญ่เพื่อศึกษาภาพรวมการเติบโต โดยใช้สัญกรณ์เชิงเส้นกำกับ (Asymptotic notations) ช่วยอธิบายลักษณะการเติบโตซึ่งง่ายสำหรับการเปรียบเทียบ

สัญกรณ์เชิงเส้นกำกับ เป็นสิ่งที่ใช้แทนความสัมพันธ์ของฟังก์ชันในแง่ของอัตราการเติบโต การใช้สัญกรณ์เชิงเส้นกำกับจะช่วยทำให้การบรรยายลักษณะฟังก์ชันทำได้ง่าย เนื่องจากไม่แสดงรายละเอียดที่ซับซ้อนของพฤติกรรมของฟังก์ชันเมื่อข้อมูลมีขนาดใหญ่ โดยสัญกรณ์เชิงเส้นกำกับที่นิยมใช้สำหรับวิเคราะห์ขั้นตอนวิธีมีด้วยกัน 3 ตัว ได้แก่ โอใหญ่ (O) โอเมก้าใหญ่ (Ω) และทีตา (Θ)

1) โอใหญ่ (O) เป็นสัญลักษณ์ที่กำหนดขอบเขตบนของเวลาที่ใช้ของขั้นตอนวิธี โดยมีนิยามดังนี้

$\mathrm{O}(g(n)) = \{f(n):$ มีค่าคงตัวบวกสองตัว คือ c และ n_0 ที่ทำให้ $0 \le f(n) \le cg(n)$ เมื่อ $n \ge n_0\}$

2) โอเมก้าใหญ่ (Ω) เป็นสัญลักษณ์ที่กำหนดขอบเขตบนของเวลาที่ใช้ของขั้นตอนวิธี โดยมีนิยามดังนี้

$\Omega(g(n)) = \{f(n):$ มีค่าคงตัวบวกสองตัว คือ c และ n_0 ที่ทำให้ $0 \le cg(n) \le f(n)$ เมื่อ $n \ge n_0\}$

3) ทีตา (Θ) เป็นสัญลักษณ์ที่กำหนดขอบเขตแน่น (Tight bound) ของเวลาที่ใช้ของขั้นตอนวิธี โดยมีนิยามดังนี้

$\Theta(g(n)) = \{f(n):$ มีค่าคงตัวบวกสามตัว คือ c_1, c_2 และ n_0 ที่ทำให้ $0 \le c_1 g(n) \le f(n) \le c_2 g(n)$ เมื่อ $n \ge n_0\}$

2.1.4 ทฤษฎีบท Master

ทฤษฎีบท Master (Master theorem) [1] เป็นทฤษฎีบทที่ใช้สำหรับหาผลเฉลยของสมการการเวียนเกิด ซึ่งผลเฉลยที่ได้จากทฤษฎีบท Master จะอยู่ในรูปของสัญญาการณ์เชิงเส้นกำกับ โดยที่สมการเวียนเกิดเมีรูปแบบดังนี้

$$\mathrm{T}(n) = a\mathrm{T}\left(\frac{n}{b}\right) + f(n)$$

ซึ่ง $a \geq 1, b > 1, f(n)$ เป็นฟังก์ชันเชิงเส้นกำกับที่มีค่าเป็นบวก

ผลเฉลยของสมการเวียนเกิดโดยใช้ทฤษฎีบท Master แบ่งออกเป็นสามกรณีดังนี้

1. ถ้า $f(n) = \mathrm{O}\left(n^{\log_b a - \varepsilon}\right)$ สำหรับบาง $\varepsilon > 0$ แล้ว $\mathrm{T}(n) = \Theta\left(n^{\log_b a}\right)$
2. ถ้า $f(n) = \Theta\left(n^{\log_b a}\right)$ แล้ว $\mathrm{T}(n) = \Theta\left(n^{\log_b a} \ln n\right)$
3. ถ้า $f(n) = \Omega\left(n^{\log_b a + \varepsilon}\right)$ สำหรับบาง $\varepsilon > 0$ และ ถ้า $af\left(\frac{n}{b}\right) \leq cf(n)$ สำหรับบาง $c < 1$ และ n มีค่าใหญ่พอแล้ว $\mathrm{T}(n) = \Theta(f(n))$

2.1.5 การวิเคราะห์เวลาการทำงานของควิกซอร์ต

การวิเคราะห์เวลาการทำงานของควิกซอร์ตสามารถหาได้จากการหาผลเฉลยของสมการเวียนเกิด จากขั้นตอนวิธีของควิกซอร์ตที่กล่าวมาข้างต้นเวลาการทำงานของควิกซอร์ตเท่ากับเวลาการทำงานที่ควิกซอร์ตใช้กับข้อมูลย่อยทางซ้ายของตัวหลักและข้อมูลย่อยทางขวาของตัวหลักบวกกับส่วนของการแบ่งกั้นข้อมูล ซึ่งเขียนเป็นสมการเวียนเกิดได้ดังนี้

$$\mathrm{T}(n) = \mathrm{T}(k) + \mathrm{T}(n-k-1) + cn$$

ซึ่ง T คือ เวลาการทำงานของควิกซอร์ต

n คือ จำนวนสมาชิกทั้งหมดในข้อมูล

k คือ จำนวนสมาชิกที่มีค่าน้อยกว่าตัวหลัก

c คือ ค่าคงตัวใดๆ

โดยที่เวลาการทำงานของควิกซอร์สามารถแบ่งเป็นสามกรณีด้วยกันดังนี้

กรณีดีที่สุดเกิดขึ้นเมื่อควิกซอร์ตแบ่งกั้นข้อมูลแล้วได้สมาชิกในข้อมูลย่อยทางซ้ายกับทางขวาของตัวหลักมีจำนวนสมาชิกเท่ากัน และตัวหลักเป็นค่ามัธยฐานของข้อมูลสำหรับทุกๆ รอบของการแบ่งกั้นข้อมูลดังรูปที่ 2.10 ซึ่งได้สมการเวียนเกิดดังนี้

$$\mathrm{T}(n) \quad = 2\mathrm{T}(n/2) + cn$$

จากกรณีที่ 2 ของทฤษฎีบท Master [1] จะได้ว่า

$$\mathrm{T}(n) \quad = \mathrm{O}(n \log n)$$

รูปที่ 2.10 การทำงานของควิกซอร์ตในกรณีที่ดีที่สุด

กรณีแย่ที่สุด เกิดขึ้นเมื่อควิกซอร์ตแบ่งกั้นข้อมูลได้สมาชิกในข้อมูลย่อยทางซ้ายของตัวหลักมีจำนวนเท่ากับศูนย์ตัวถ้าตัวหลักเป็นค่าน้อยที่สุดของข้อมูล หรือสมาชิกในข้อมูลย่อยทางขวาของตัวหลักมีจำนวนเท่ากับศูนย์ตัวถ้าตัวหลักเป็นค่ามากที่สุดของข้อมูลสำหรับทุกๆ รอบของการแบ่งกั้นข้อมูล การวิเคราะห์เวลาการทำงานของควิกซอร์ตกรณีแย่ที่สุดสมมติให้ตัวหลักเป็นค่าน้อยที่สุดซึ่งแสดงดังรูปที่ 2.11 ซึ่งได้สมการเวียนเกิดดังนี้

$$\mathrm{T}(n) \quad = \mathrm{T}(0) + \mathrm{T}(n-1) + cn$$

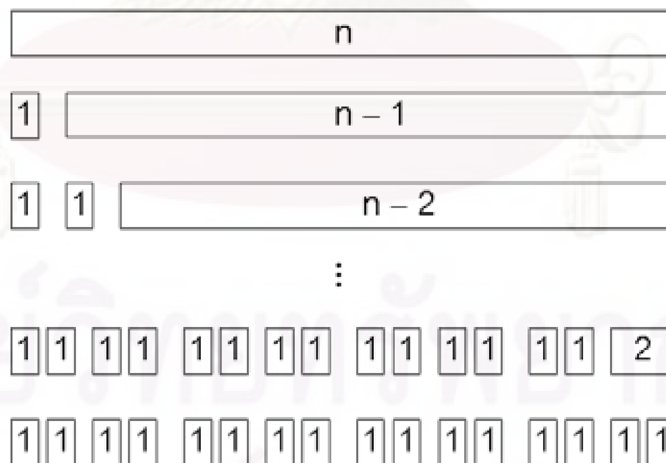
ซึ่ง $\mathrm{T}(0)$ เป็นเวลาการทำงานของควิกซอร์ตที่ไม่มีสมาชิก ดังนั้น $\mathrm{T}(0) = 0$ และจะได้

$$\mathrm{T}(n) \quad = \mathrm{T}(n-1) + cn$$

$$\begin{aligned} \mathrm{T}(n-1) &= \mathrm{T}(n-2) + c(n-1) \\ \mathrm{T}(n-2) &= \mathrm{T}(n-3) + c(n-2) \\ &\vdots \\ \mathrm{T}(2) &= \mathrm{T}(1) + c2 \end{aligned}$$

เขียน $\mathrm{T}(n)$ ให้อยู่ในรูปของ n ได้ดังนี้

$$\begin{aligned} \mathrm{T}(n) &= \mathrm{T}(1) + c\left[2 + 3 + \ldots + (n-1) + n\right] \\ \mathrm{T}(n) &= \mathrm{T}(1) + c\left(\frac{n^2 + n - 2}{2}\right) \\ \mathrm{T}(n) &= \mathrm{O}\left(n^2\right) \end{aligned}$$



รูปที่ 2.11 การทำงานของควิกซอร์ตในกรณีแย่ที่สุดเมื่อตัวหลักเป็นค่าน้อยที่สุด

กรณีเฉลี่ย พิจารณาโดยกำหนดให้การแบ่งกั้นข้อมูลในกรณีต่างๆ มีความน่าจะเป็นที่จะเกิดขึ้นเท่าๆ กัน และเวลาการทำงานของข้อมูลย่อยทางซ้ายของตัวหลักเท่ากับข้อมูลย่อยทางขวาของตัวหลัก ซึ่งได้สมการเวียนเกิดดังนี้

$$\mathrm{T}(n) = 2\left[\frac{\mathrm{T}(0)+\mathrm{T}(1)+\ldots+\mathrm{T}(n-2)+\mathrm{T}(n-1)}{n}\right]+cn$$

นำ n มาคูณตลอดทั้งสมการจะได้

$$n\mathrm{T}(n) = 2\left[\mathrm{T}(0)+\mathrm{T}(1)+\ldots+\mathrm{T}(n-2)+\mathrm{T}(n-1)\right]+cn^2 \qquad (1)$$

เมื่อ $n>1$ แทนค่า $n=n-1$ ลงในสมการที่ (1) จะได้

$$(n-1)\mathrm{T}(n-1) = 2\left[\mathrm{T}(0)+\mathrm{T}(1)+\ldots+\mathrm{T}(n-3)+\mathrm{T}(n-2)\right]+c(n-1)^2 \qquad (2)$$

นำสมการ (1) – (2) จะได้

$$n\mathrm{T}(n)-(n-1)\mathrm{T}(n-1) = 2 \quad (n\mp 1)+2cn-c$$

จัดรูปสมการใหม่และตัดค่าคงที่ออก ดังนั้นจะได้

$$n\mathrm{T}(n) = (n+1)\mathrm{T}(n-1)+2cn$$

นำ $\frac{1}{n(n+1)}$ มาคูณตลอดทั้งสมการจะได้

$$\frac{\mathrm{T}(n)}{n+1} = \frac{\mathrm{T}(n-1)}{n}+\frac{2c}{n+1}$$

$$\frac{\mathrm{T}(n-1)}{n} = \frac{\mathrm{T}(n-2)}{n-1}+\frac{2c}{n}$$

$$\frac{\mathrm{T}(n-2)}{n-1} = \frac{\mathrm{T}(n-3)}{n-2}+\frac{2c}{n-1}$$

$$\vdots$$

$$\frac{\mathrm{T}(2)}{3} = \frac{\mathrm{T}(1)}{2}+\frac{2c}{3}$$

ศูนย์วิทยทรัพยากร
จุฬาลงกรณ์มหาวิทยาลัย

เขียน $\frac{T(n)}{n+1}$ ให้อยู่ในรูปของ n ได้ดังนี้

$$\frac{T(n)}{n+1} = \frac{T(1)}{2} + 2c\left(\frac{1}{3}+\frac{1}{4}+\ldots+\frac{1}{n}+\frac{1}{n+1}\right)$$
$$= \frac{T(1)}{2} + 2c\left(1+\frac{1}{2}+\frac{1}{3}+\ldots+\frac{1}{n}+\frac{1}{n+1}\right) - 3c$$

ตัด $T(1)$ และ c ซึ่งเป็นค่าคงที่ออก และอนุกรมฮาร์มอนิกสามารถประมาณเป็น $\ln(n)$ ดังนั้นจะได้

$$T(n) = O(n \log n)$$

2.1.6 มัธยฐานของสามตัว

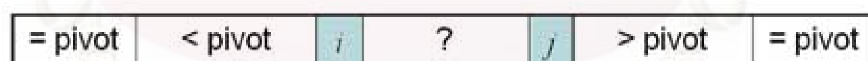
มัธยฐานของสามตัว (Median of three) [4] เป็นวิธีการเลือกตัวหลักจากการประมาณค่ามัธยฐานของข้อมูล ซึ่งค่าประมาณมัธยฐานของข้อมูลหาได้จากค่ามัธยฐานจากสมาชิก 3 ตำแหน่ง คือ ตำแหน่งแรก ตำแหน่งกลางและตำแหน่งท้าย โดยทำการเปรียบเทียบสมาชิกตำแหน่งแรกกับสมาชิกตำแหน่งกลาง ถ้าสมาชิกตำแหน่งแรกมีค่ามากกว่าสมาชิกตำแหน่งกลางแล้วจะสลับที่สมาชิกตำแหน่งแรกกับตำแหน่งกลาง จากนั้นทำการเปรียบเทียบสมาชิกตำแหน่งแรกกับสมาชิกตำแหน่งท้าย ถ้าสมาชิกตำแหน่งแรกมีค่ามากกว่าสมาชิกตำแหน่งท้ายแล้วจะสลับที่สมาชิกตำแหน่งแรกกับตำแหน่งท้าย สุดท้ายทำการเปรียบเทียบสมาชิกตำแหน่งกลางกับตำแหน่งท้าย ถ้าสมาชิกตำแหน่งกลางมีค่ามากกว่าสมาชิกตำแหน่งท้ายแล้วสลับที่สมาชิกตำแหน่งกลางกับตำแหน่งท้าย ซึ่งจะได้สมาชิกตำแหน่งกลางเป็นค่ามัธยฐานของสมาชิกสามตัว วิธีการเลือกตัวหลักแบบนี้จะทำให้การประมวลผลของควิกซอร์ตมีประสิทธิภาพดีขึ้น เนื่องจากตัวหลักที่เลือกมีค่าใกล้เคียงกับค่ามัธยฐานของข้อมูล จากหลักการข้างต้นสามารถเขียนเป็นรหัสคำสั่งเทียมได้ดังนี้

```
Procedure Median of three (A[ ], left, middle, right)
    IF (A[left] > A[middle]) THEN
        Swap A[left] and A[middle];
    ENDIF
```

```
    IF (A[left] > A[right]) THEN
        Swap A[left] and A[right];
    ENDIF
    IF (A[middle] > A[right]) THEN
        Swap A[middle] and A[right];
    ENDIF
END Median of three
```

2.1.7 การแบ่งกั้นข้อมูลแบบสปลิตเอ็น

การแบ่งกั้นข้อมูลแบบสปลิตเอ็น (Split-end partition) [7] ถูกเสนอขึ้นโดย Bentley และ Mcilroy ในปี 1993 เป็นวิธีการแบ่งกั้นข้อมูลที่สามารถจัดการกับสมาชิกที่มีค่าซ้ำกันได้ ถ้าสมาชิกในข้อมูลมีค่าเท่ากันหมดทุกตัวแล้วการแบ่งกั้นข้อมูลแบบสปลิตเอ็นจะหยุดการทำซ้ำทันที ในขณะที่การแบ่งกั้นข้อมูลแบบธรรมดาจะต้องแบ่งกั้นข้อมูลไปจนกระทั่งไม่สามารถแบ่งกั้นข้อมูลได้อีกจึงหยุดการทำซ้ำ การแบ่งแบบสปลิตเอ็นนั้นเหมาะสำหรับข้อมูลที่มีสมาชิกซ้ำกันเป็นจำนวนมาก โดยการแบ่งกั้นข้อมูลแบบสปลิตเอ็นจะทำการตรวจหาสมาชิกอื่นที่มีค่าเท่ากับตัวหลักระหว่างที่ทำการแบ่งกั้นข้อมูล ถ้าสมาชิกที่ตัวชี้ i ชี้อยู่มีค่าเท่ากับตัวหลักสมาชิกนั้นจะถูกย้ายไปอยู่ทางขอบซ้ายของข้อมูล และถ้าสมาชิกที่ตัวชี้ j ชี้อยู่มีค่าเท่ากับตัวหลักสมาชิกนั้นจะถูกย้ายไปอยู่ทางขอบขวาของข้อมูล ซึ่งแสดงดังรูปที่ 2.12



2.12 การแบ่งกั้นข้อมูลแบบสปลิตเอ็น

หลังจบการแบ่งกั้นข้อมูลแบบสปลิตเอ็นจะได้ตำแหน่งที่ถูกต้องของตัวหลัก จากนั้นทำการย้ายสมาชิกที่มีค่าเท่ากับตัวหลักซึ่งถูกเก็บไว้ที่ขอบทั้งสองข้างให้ไปอยู่ในตำแหน่งที่ถูกต้องทำให้ได้ข้อมูลย่อยสามกลุ่ม คือ กลุ่มข้อมูลย่อยที่สมาชิกมีค่าน้อยตัวหลัก กลุ่มข้อมูลย่อยที่สมาชิกมีค่าเท่ากับตัวหลัก และกลุ่มข้อมูลย่อยที่สมาชิกที่มีค่ามากกว่าตัวหลัก ซึ่งแสดงดังรูปที่ 2.13

< pivot	= pivot	> pivot

2.13 ข้อมูลย่อยสามกลุ่มของการแบ่งกั้นข้อมูลแบบสปลิตเอ็น

จากหลักการข้างต้นสามารถเขียนเป็นรหัสคำสั่งได้ดังนี้ ในที่นี้กำหนดให้ตัวหลักเป็นสมาชิกตำแหน่งขวาของข้อมูลเสมอ

```
Function Split-end partition (A[ ], left, right)
    i = pl = left;
    j = pr = right – 1;
    pivot = A[right];

    LOOP
        WHILE (A[i] ≤ pivot AND i ≤ j) DO
            IF (A[i] == pivot) THEN
                Swap A[i] and A[pl];
                pl = pl + 1;
            ENDIF
            i = i + 1;
        ENDWHILE
        WHILE (A[j] ≥ pivot AND i ≤ j) DO
            IF (A[j] == pivot) THEN
                Swap A[j] and A[pr];
                pr = pr – 1;
            ENDIF
            j = j – 1;
        ENDWHILE
        IF (i > j) THEN
            Exit LOOP;
        ENDIF
        Swap A[i] and A[j];
    END LOOP

    Move all elements which equal to pivot to the correct positions.
    Return indices of pivot.
}
```

ศูนย์วิทยทรัพยากร
จุฬาลงกรณ์มหาวิทยาลัย

ตัวอย่างการทำงานของควิกซอร์ตด้วยการแบ่งกั้นข้อมูลแบบสปลิตเอ็น โดยกำหนดข้อมูลดังนี้ : 1, 8, 6, 3, 9, 6, 4, 6, 6

เริ่มต้นเลือกตัวหลักจากสมาชิกตัวสุดท้าย คือ 6 สำหรับการแบ่งกั้นข้อมูล แล้วกำหนดให้ i กับ ipL เป็นตัวชี้จากตำแหน่งแรก และ j กับ ipR เป็นตัวชี้จากตำแหน่งรองสุดท้าย ซึ่งตัวชี้ ipL และ ipR เป็นตัวชี้ที่ใช้สำหรับเก็บสมาชิกที่มีค่าเท่ากับตัวหลักทางขอบซ้ายและขอบขวาตามลำดับ

1	8	6	3	9	6	4	6	<u>6</u>
i, ipL							*j, ipR*	

ทำการเปรียบเทียบสมาชิกที่ตัวชี้ i กับตัวหลัก โดยขยับตัวชี้ i มาทางด้านขวาจนกระทั่งเจอสมาชิกที่มีค่ามากกว่า 6 พร้อมทั้งตรวจหาสมาชิกที่มีค่าเท่ากับตัวหลัก ซึ่งตัวชี้ i หยุดที่สมาชิกตำแหน่งที่สองซึ่งมีค่าเท่ากับ 8

1	8	6	3	9	6	4	6	<u>6</u>
ipL	*i*						*j, ipR*	

ทำการเปรียบเทียบสมาชิกที่ตัวชี้ j กับตัวหลัก โดยขยับตัวชี้ j มาทางด้านซ้ายจนกระทั่งเจอสมาชิกที่มีค่าน้อยกว่า 6 พร้อมทั้งตรวจหาสมาชิกที่มีค่าเท่ากับตัวหลัก ซึ่งพบว่าสมาชิกตำแหน่งที่แปดมีค่าเท่าตัวหลักจึงสลับที่สมาชิกตำแหน่งที่แปดกับสมาชิกที่ตัวชี้ ipR ชี้อยู่เผอิญว่าเป็นตัวเดียวกัน และขยับตัวชี้ ipR มาทางซ้ายหนึ่งช่อง โดยตัวชี้ j หยุดที่สมาชิกตำแหน่งที่เจ็ดซึ่งมีค่าเท่ากับ 4

1	8	6	3	9	6	4	6	<u>6</u>
ipL	*i*					*j, ipR*		

ทำการเปรียบเทียบตัวชี้ i กับตัวชี้ j โดยตัวชี้ i ชี้ที่ตำแหน่งที่สองซึ่งน้อยกว่าตัวชี้ j ชี้ที่ตำแหน่งที่เจ็ด ดังนั้นทำการสลับที่สมาชิกระหว่างตัวชี้ i กับตัวชี้ j

1	4	6	3	9	6	8	6	<u>6</u>
ipL	*i*					*j, ipR*		

ศูนย์วิทยทรัพยากร
จุฬาลงกรณ์มหาวิทยาลัย

ทำการเปรียบเทียบสมาชิกที่ตัวชี้ i กับตัวหลัก โดยขยับตัวชี้ i มาทางด้านขวาจนกระทั่งเจอสมาชิกที่มีค่ามากกว่า 6 กับตัวหลักพร้อมทั้งตรวจหาสมาชิกที่มีค่าเท่ากับตัวหลัก ซึ่งพบว่าสมาชิกตำแหน่งที่สามมีค่าเท่าตัวหลักจึงสลับที่สมาชิกตำแหน่งที่สามกับสมาชิกที่ตัวชี้ ipL ชี้อยู่ และขยับตัวชี้ ipL มาทางขวาหนึ่งช่อง โดยตัวชี้ i หยุดที่สมาชิกตำแหน่งที่ห้าซึ่งมีค่าเท่ากับ 9

6	4	1	3	9	6	8	6	<u>6</u>
	ipL			*i*		*j, ipR*		

ทำการเปรียบเทียบสมาชิกที่ตัวชี้ j กับตัวหลัก โดยขยับตัวชี้ j มาทางด้านซ้ายจนกระทั่งเจอสมาชิกที่มีค่าน้อยกว่า 6 พร้อมทั้งตรวจหาสมาชิกที่มีค่าเท่ากับตัวหลัก ซึ่งพบว่าสมาชิกตำแหน่งที่เจ็ดมีค่าเท่าตัวหลักจึงสลับที่สมาชิกตำแหน่งที่เจ็ดกับสมาชิกที่ตัวชี้ ipR ชี้อยู่ และขยับตัวชี้ ipR มาทางซ้ายหนึ่งช่อง โดยตัวชี้ j หยุดที่สมาชิกตำแหน่งที่สี่ซึ่งมีค่าเท่ากับ 3

6	4	1	3	9	8	6	6	<u>6</u>
	ipL		*j*	*i*	*ipR*			

ทำการเปรียบเทียบตัวชี้ i กับตัวชี้ j โดยตัวชี้ i ชี้ที่ตำแหน่งที่ 5 ซึ่งมากกว่าตัวชี้ j ชี้ที่ตำแหน่งที่ 4 เป็นอันจบการแบ่งกั้นข้อมูล จากนั้นทำการย้ายสมาชิกที่มีค่าเท่ากับตัวหลักไปอยู่ในตำแหน่งที่ถูกต้อง โดยย้ายสมาชิกที่มีค่าเท่ากับตัวหลักจากขอบซ้ายกับสมาชิกที่ตัวชี้ j และขยับตัวชี้ j ไปทางด้านซ้ายหนึ่งช่อง และสมาชิกที่มีค่าเท่ากับตัวหลักจากขอบขวากับสมาชิกที่ i และขยับตัวชี้ i ไปทางขวาหนึ่งช่อง ทำซ้ำจนกระทั่งย้ายสมาชิกที่มีค่าเท่ากับหลักครบทุกตัว

3	4	1	<u>6</u>	<u>6</u>	<u>6</u>	<u>6</u>	8	9

เมื่อถึงขั้นตอนนี้สมาชิกทุกตัวที่มีค่าเท่ากับตัวหลักจะอยู่ ณ ตำแหน่งที่ถูกต้อง และได้ข้อมูลย่อยสามกลุ่ม คือ ข้อมูลย่อยทางซ้ายที่สมาชิกมีค่าน้อยกว่าตัวหลัก ข้อมูลย่อยที่สมาชิกมีค่าเท่ากับตัวหลัก และข้อมูลย่อยทางขวาที่สมาชิกมีค่ามากกว่าตัวหลัก จากนั้นทำการแบ่งข้อมูลซ้ำกับข้อมูลทางซ้ายและข้อมูลย่อยทางขวาจนกระทั่งไม่สามารถแบ่งข้อมูลได้อีกควิกซอร์ตก็จะหยุด ผลลัพธ์ที่ได้ก็คือข้อมูลที่เรียงลำดับจากน้อยไปมาก

2.2 งานวิจัยที่เกี่ยวข้อง

ในหัวข้อนี้จะกล่าวถึงเทคนิคต่างๆ ที่นำมาใช้ปรับปรุงประสิทธิภาพการทำงานของควิกซอร์ตที่ผู้วิจัยได้ไปศึกษามาจากงานวิจัยดังนี้

ในปี 1961, Hoare [2] เป็นผู้เสนอวิธีของควิกซอร์ตเป็นคนแรก โดยให้เลือกตัวหลักมาจากการสุ่มสมาชิกในข้อมูลที่ต้องการเรียงลำดับ

ในปี 1965, Scowen [3] เสนอ Quickersort ที่มีหลักการทำงานเหมือนควิกซอร์ต แต่ให้ใช้ตัวหลักจากข้อมูลที่ตำแหน่งกลางของข้อมูล และถ้ากลุ่มข้อมูลย่อยมีสมาชิกสองตัวก็จะเปรียบเทียบธรรมดา ซึ่งการเลือกตัวหลักแบบนี้จะมีประสิทธิภาพเมื่อข้อมูลมีการเรียงลำดับที่ถูกต้องอยู่แล้ว เนื่องจากการเลือกตัวหลักแบบนี้จะเป็นค่ากลางของชุดข้อมูลซึ่งทำให้การทำงานของควิกซอร์ตเป็นกรณีที่ดีที่สุด

ในปี 1969, Singleton [4] เสนอวิธีการเลือกตัวหลักจากการประมาณค่ามัธยฐานของข้อมูล โดยค่าประมาณมัธยฐานของข้อมูลหามาจากค่ามัธยฐานจากสมาชิก 3 ตำแหน่ง คือ ตำแหน่งแรก ตำแหน่งกลางและตำแหน่งท้าย ซึ่งวิธีการนี้เรียกว่ามัธยฐานของสาม วิธีการเลือกตัวหลักแบบนี้จะทำให้การประมวลของควิกซอร์ตมีประสิทธิภาพดีขึ้น เนื่องจากตัวหลักที่เลือกมีโอกาสสูงที่จะค่าใกล้เคียงกับค่ามัธยฐานของข้อมูลขณะนั้น

ในปี 1978, Sedgewick [5] เสนอควิกซอร์ตที่เหมาะสำหรับการใช้งานบนคอมพิวเตอร์ โดยแนะนำให้ใช้ควิกซอร์ตที่ไม่มีการเรียกใช้ฟังก์ชันเวียนเกิด เลือกตัวหลักจากวิธีมัธยฐานของสาม และใช้การเรียงลำดับแบบแทรกแทนควิกซอร์ตเมื่อข้อมูลย่อยมีสมาชิกน้อยกว่า 9 และ 10 ตัว ซึ่งวิธีการนี้สามารถเพิ่มประสิทธิภาพในการประมวลผลของควิกซอร์ตได้ประมาณ 20 เปอร์เซ็นต์

ในปี 1980, Cook และ Kim [13] เสนอวิธีการเรียงลำดับที่เหมาะสมกับข้อมูลที่เกือบเรียงลำดับ โดยไล่พิจารณาสมาชิกที่ไม่เรียงลำดับในข้อมูลแล้วนำออกไปเก็บสะสมไว้ในข้อมูลใหม่ ถ้าข้อมูลใหม่มีสมาชิกมากกว่า 30 ตัว จะเรียกใช้ควิกซอร์ต ไม่เช่นนั้นจะเรียกใช้การเรียงลำดับแบบแทรก เมื่อได้ข้อมูลที่เรียงลำดับทั้งสองชุดแล้วจึงนำมาผสานกัน

ในปี 1983, Motzkin [14] เสนอ Meansort ที่มีหลักการทำงานเหมือนควิกซอร์ต แต่มีการคำนวณค่าเฉลี่ยของแต่ละกลุ่มข้อมูลขณะที่ทำการแบ่งกั้นข้อมูลเพื่อใช้เป็นตัวหลักแทนการเลือกตัวหลักจากสมาชิกในข้อมูล วิธีการเลือกตัวหลักแบบนี้จะทำให้เกิดกรณีแย่ที่สุดของควิกซอร์ตได้น้อยครั้ง เนื่องจากตัวหลักเลือกมาจากค่าเฉลี่ยของข้อมูลทำให้เป็นไปได้น้อยมากที่จะได้ตัวหลักเป็นค่ามากที่สุดหรือค่าน้อยที่สุด

ในปี 1985, Wainwright [6] เสนอ Bsort ที่มีหลักการทำงานเหมือนควิกซอร์ต แต่มีการตรวจสอบการเรียงลำดับของข้อมูลโดยอาศัยหลักการทำงานของการเรียงลำดับแบบฟอง ซึ่งถ้าสมาชิกที่ทำการตรวจสอบไม่เรียงลำดับก็จะสลับที่กัน โดย Bsort จะประมวลผลได้ดีสำหรับข้อมูลเกือบเรียงและเกือบเรียงลำดับแบบย้อนกลับ

ในปี 1987, Wainwright [15] เสนอ QSORTE ซึ่งเป็นขั้นตอนวิธีที่ถูกปรับปรุงมาจาก Bsort โดยตัดขั้นตอนการสลับที่สำหรับสมาชิกที่ไม่เรียงลำดับออก และปรับเปลี่ยนวิธีการตรวจสอบการเรียงลำดับของข้อมูล

ในปี 1993, Bentley และ Mcilroy [7] เสนอวิธีการเลือกตัวหลักจากมัธยฐานของสามหรือมัธยฐานของเก้าโดยขึ้นกับจำนวนสมาชิกในข้อมูล และเสนอวิธีการแบ่งกั้นข้อมูลโดยย้ายไปด้านท้ายซึ่งเป็นวิธีการแบ่งกั้นข้อมูลที่สามารถจัดการกับข้อมูลที่มีสมาชิกซ้ำกันได้ซึ่งการแบ่งกั้นข้อมูลนี้จะแบ่งข้อมูลออกเป็นสามกลุ่ม คือ กลุ่มที่สมาชิกมีค่าน้อยกว่าตัวหลัก กลุ่มที่สมาชิกมีค่าเท่ากับตัวหลัก และกลุ่มที่สมาชิกมีค่ามากกว่าตัวหลัก

ในปี 1996, Sarwar และคณะ [16] เสนอวิธีการเลือกตัวหลักจากมัธยฐานของสามหรือมัธยฐานของห้าหรือมัธยฐานของเก้าหรือมัธยฐานของสิบเจ็ดโดยขึ้นกับจำนวนสมาชิกในข้อมูล ซึ่งวิธีการเลือกตัวหลักแบบนี้จะมีประสิทธิภาพดีกว่าการเลือกตัวหลักโดยใช้วิธีมัธยฐานของสามเพียงอย่างเดียว

ในปี 1998, Hossain และคณะ [17] เสนอวิธีการปรับปรุงควิกซอร์ต โดยการแบ่งข้อมูลออกเป็น $\lceil \sqrt{n} \rceil$ ส่วน ซึ่งแต่ละส่วนจะมีจำนวนสมาชิกไม่เกิน $\lceil \sqrt{n} \rceil$ ตัว โดยที่ n คือ จำนวนสมาชิกของข้อมูล แล้วนำข้อมูลแต่ละส่วนมาเรียงลำดับด้วยควิกซอร์ต จากนั้นนำข้อมูลที่เรียงลำดับแล้วแต่ละส่วนมาผสานกัน

ในปี 2006, Edmondson [18] เสนอ M Pivot Sort ที่มีหลักการทำงานเหมือน ควิกซอร์ต โดยจะใช้ตัวหลักหลายตัวแทนการใช้ตัวหลักเพียงตัวเดียว และถ้าข้อมูลย่อยมีขนาดน้อยกว่าหรือเท่ากับ 15 จะใช้การเรียงลำดับแบบแทรกแทน M Pivot Sort

ในปี 2007, Jidol และคณะ [19] เสนอวิธีการปรับปรุง ควิกซอร์ตโดยใช้ผลต่างสืบเนื่องสำหรับตรวจสอบการเรียงลำดับของข้อมูล โดยผลต่างสืบเนื่องสามารถตรวจสอบได้ทั้งการเรียงลำดับแบบธรรมดาและการเรียงลำดับแบบย้อนกลับ

ศูนย์วิทยทรัพยากร
จุฬาลงกรณ์มหาวิทยาลัย

บทที่ 3

การปรับปรุงควิกซอร์ตด้วยตัวนำหน้าและตัวตามหลังของตัวหลัก

งานวิจัยนี้เสนอการปรับปรุงควิกซอร์ตด้วยตัวนำหน้าตัวหลักและตัวตามหลังตัวหลักซึ่งเป็นขั้นตอนวิธีสำหรับปรับปรุงควิกซอร์ตให้มีประสิทธิภาพดีขึ้นสำหรับข้อมูลที่มีสมาชิกซ้ำกัน โดยวิธีการนี้เน้นการลดจำนวนครั้งที่ใช้ฟังก์ชันเวียนเกิดซึ่งใช้ประโยชน์จากสมาชิกที่อยู่ตำแหน่งนำหน้าตัวหลักเรียกว่า ตัวนำหน้าตัวหลัก (PD) และสมาชิกที่อยู่ตำแหน่งตามหลังตัวหลักเรียกว่า ตัวตามหลังตัวหลัก (SC) ร่วมกับการแบ่งกั้นข้อมูลแบบสปลิตเอ็น (Split-end partition) บทนี้อธิบายที่มาของการปรับปรุงควิกซอร์ตด้วยตัวนำหน้าตัวหลักและตัวตามหลังตัวหลักซึ่งมีรายละเอียดดังต่อไปนี้

3.1 ขั้นตอนวิธีควิกซอร์ตด้วยตัวนำหน้าและตัวตามหลังของตัวหลัก

ควิกซอร์ตด้วยตัวนำหน้าและตัวตามหลังของตัวหลัก (PSQuicksort) ใช้มัธยฐานของสามตัว (Median of three) [4] สำหรับกำหนดตัวหลักซึ่งได้ค่ามัธยฐานของสามตัวเป็นตัวหลัก และใช้สมาชิกที่มีค่าน้อยกว่าตัวหลักกับสมาชิกที่มีค่ามากกว่าตัวหลักเป็นตัวแทนของตัวนำหน้าตัวหลักและตัวตามหลังตัวหลัก โดยสมาชิกที่มีค่าน้อยกว่ามัธยฐานเป็นตัวแทนของตัวนำหน้าตัวหลักเรียกว่า ตัวนำหน้าตัวหลักเทียม (p-PD) ส่วนสมาชิกที่มีค่ามากกว่ามัธยฐานถูกเป็นตัวแทนของตัวตามหลังตัวหลักเรียกว่า ตัวตามหลังตัวหลักเทียม (p-SC)

การแบ่งกั้นข้อมูลของ PSQuicksort จะนำตัวนำหน้าตัวหลักเทียมและตัวตามหลังตัวหลักเทียมมาช่วยในการแบ่งกั้นข้อมูลซึ่งค่าของทั้งสองตัวจะถูกปรับเปลี่ยนขณะที่ทำการแบ่งกั้นข้อมูลหลังจบการแบ่งกั้นข้อมูลจึงได้ค่าที่แท้จริงของตัวนำหน้าตัวหลักและตัวตามหลังตัวหลัก จากนั้นย้ายตัวนำหน้าตัวหลัก ตัวหลักและตัวตามหลังตัวหลักมาอยู่ในตำแหน่งที่ถูกต้องซึ่งทำให้ทราบตำแหน่งที่ถูกต้องของสมาชิกสามตัวสำหรับทุกรอบของการแบ่งกั้นข้อมูล และได้ข้อมูลย่อยสามกลุ่ม คือ ข้อมูลย่อยทางซ้ายของตัวหลักซึ่งสมาชิกมีค่าน้อยกว่าตัวนำหน้าตัวหลัก ข้อมูลย่อยที่เรียงลำดับแล้วซึ่งเก็บตัวนำหน้าตัวหลัก ตัวหลักและตัวตามหลังตัวหลัก และข้อมูลย่อยทางขวาของตัวหลักซึ่งสมาชิกมีค่ามากกว่าตัวตามหลังตัวหลัก ซึ่งแสดงดังรูปที่ 3.1 จากนั้นทำซ้ำขั้นตอนข้างต้นจนกระทั่งข้อมูลย่อยมีจำนวนสมาชิกน้อยกว่าหรือเท่ากับ M ตัว แล้วจะใช้การเรียงลำดับ

แบบแทรกแทน PSQuicksort จากงานวิจัยของ Sedgewick [5] เสนอให้ใช้ M = 9 และจากหลักการข้างต้นสามารถเขียนเป็นรหัสคำสั่งเทียมได้ดังนี้

รูปที่ 3.1 ลักษณะข้อมูลย่อยหลังการแบ่งกั้นข้อมูลของ PSQuicksort

```
Function PSQuicksort (A[ ], left, right)
    IF size of list A ≤ M THEN
        Call InsertionSort (A, left, right);
    ELSE
        mid = (left + right)/2;
        Select pivot with Median of three (A, left, mid, right);
        {ind_L, ind_R} = PSpartition (A, left, mid, right);
        Recursively call PSQuicksort (A, left, ind_L);
        Recursively call PSQuicksort (A, ind_R, right);
    ENDIF
End PSQuicksort
```

มัธยฐานของสามตัวที่ใช้ใน PSQsuicksort มีขั้นตอนวิธีดังนี้ ถ้าสมาชิกตัวซ้ายมีค่ามากกว่าสมาชิกตัวกลางแล้วสลับที่สมาชิกตัวซ้ายกับตัวกลาง ถ้าสมาชิกตัวซ้ายมีค่ามากกว่าสมาชิกตัวขวาแล้วสลับที่สมาชิกตัวซ้ายกับตัวขวา และถ้าสมาชิกตัวขวามีค่ามากกว่าสมาชิกตัวกลางแล้วสลับที่สมาชิกตัวขวากับตัวกลาง จากขั้นตอนข้างต้นจะได้ว่า สมาชิกตัวซ้ายเป็นตัวนำหน้าตัวหลักเทียม สมาชิกตัวกลางเป็นตัวตามหลังตัวหลักเทียม และสมาชิกตัวขวาเป็นตัวหลัก ซึ่งสามารถเขียนเป็นรหัสคำสั่งเทียมได้ดังนี้

```
Function Median of three (A[ ], left, middle, right){
    IF (A[left] > A[middle]) THEN
        Swap A[left] and A[middle];
    ENDIF
    IF (A[left] > A[right]) THEN
        Swap A[left] and A[right];
    ENDIF
    IF (A[right] > A[mid]) THEN
        Swap A[middle] and A[right];
    ENDIF
End Median of three
```

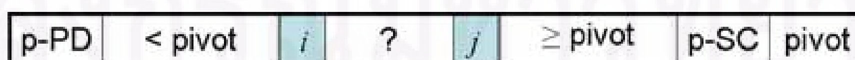
ขั้นตอนวิธีการเรียงลำดับแบบแทรกที่ใช้ในงานวิจัยนี้มาจาก [20] ซึ่งสามารถเขียนเป็นรหัสเทียมได้ดังนี้

```
Function Insertion Sort (A[ ], left, right)
    FOR (i = left + 1; i <= right; i++) DO
        Temp = A[i];
        FOR (j = i; j > left AND A[j – 1] > Temp; j--) DO
            A[j] = A[j – 1];
        ENDFOR
        A[j] = Temp;
    ENDFOR
End Insertion Sort
```

ขั้นตอนวิธีการแบ่งกั้นข้อมูลด้วยตัวนำหน้าและตัวตามหลังของตัวหลักที่ผู้วิจัยพัฒนามีสองรูปแบบคือ

3.1.1 การแบ่งกั้นข้อมูลด้วยตัวนำหน้าและตัวตามหลังของตัวหลักที่ใช้การค้นหาโดยตรง

การแบ่งกั้นข้อมูลด้วยตัวนำหน้าและตัวตามหลังของตัวหลักที่ใช้การค้นหาโดยตรง (PSpartition_direct) เป็นการหาค่ามากที่สุดของข้อมูลย่อยทางซ้ายของตัวหลัก และหาค่ามากที่สุดของข้อมูลย่อยทางขวาของตัวหลักในขณะที่ทำการแบ่งกั้นข้อมูล โดยเพิ่มการเปรียบเทียบหนึ่งครั้งในการวนซ้ำที่ตัวชี้ i เพื่อหาสมาชิกที่มีค่ามากกว่าตัวนำหน้าตัวหลักเทียม ถ้าสมาชิกนั้นมีค่ามากกว่าตัวนำหน้าตัวหลักเทียมแล้วมันจะถูกย้ายมาที่ขอบซ้ายของข้อมูล และเพิ่มการเปรียบเทียบหนึ่งครั้งในการวนซ้ำที่ตัวชี้ j เพื่อหาสมาชิกที่มีค่าน้อยกว่าตัวตามหลังตัวหลักเทียม ถ้าสมาชิกนั้นมีค่าน้อยกว่าตัวตามหลังตัวหลักเทียมแล้วมันจะถูกย้ายมาที่ขอบขวาของข้อมูล ซึ่งแสดงดังรูปที่ 3.2 หลังจบการแบ่งกั้นข้อมูล ตัวนำหน้าตัวหลักเทียมจะเป็นตัวนำหน้าตัวหลัก และตัวตามหลังตัวหลักเทียมจะเป็นตัวตามหลังตัวหลัก จากนั้นจึงย้ายตัวนำหน้าตัวหลัก ตัวหลักและตัวตามหลังตัวหลักไปอยู่ในตำแหน่งที่ถูกต้อง จากหลักการข้างต้นสามารถเขียนเป็นรหัสคำสั่งเทียมได้ดังนี้



รูปที่ 3.2 การแบ่งกั้นข้อมูลด้วยตัวนำหน้าและตัวตามหลังของตัวหลักที่ใช้การค้นหาโดยตรง

```
Function PSpartition_direct (A[ ], left, mid, right)
    pivot = A[right];
    Swap A[mid] and A[right – 1];
    i = left + 1;
    j = right – 2;
    p-PD = A[left];
    p-SC = A[right – 1];

    LOOP
        WHILE (A[i] < pivot) DO
            IF (A[i] > p-PD) THEN
                Swap A[i] and A[left];
            ENDIF
            i = i + 1;
        ENDWHILE
        WHILE (A[j] ≥ pivot) DO
            IF (A[j] < p-SC) THEN
                Swap A[j] and A[right – 1];
            ENDIF
            j = j – 1;
        ENDWHILE
        IF (i ≥ j) THEN
            Exit LOOP;
        ENDIF
        Swap A[i] and A[j];
    ENDLOOP

    Move PD, pivot and SC to the correct positions.
    Return index of PD and SC.
End PSpartition_direct
```

3.1.2 การแบ่งกั้นข้อมูลด้วยตัวนำหน้าและตัวตามหลังของตัวหลักที่ใช้การเรียงลำดับแบบฟอง

การแบ่งกั้นข้อมูลด้วยตัวนำหน้าและตัวตามหลังของตัวหลักที่ใช้การเรียงลำดับแบบฟอง (PSpartition_bubble) เป็นการนำหลักการทำงานของการเรียงลำดับแบบฟองมาใช้ดันสมาชิกที่มีค่ามากที่สุดของข้อมูลย่อยทางซ้ายของตัวหลักให้ไปอยู่ติดกับตัวหลัก และดันสมาชิกที่มีค่าน้อยที่สุดของข้อมูลย่อยทางขวาของตัวหลักให้ไปอยู่ติดกับตัวหลักในขณะที่ทำการแบ่งกั้นข้อมูล โดยเพิ่มการเปรียบเทียบหนึ่งครั้งในการวนซ้ำที่ตัวชี้ i เพื่อปรับค่าและดันตัวนำหน้าตัวหลักเทียมให้ไปอยู่ในตำแหน่งที่ถูกต้อง และเพิ่มการเปรียบเทียบหนึ่งครั้งในการวนซ้ำที่ตัวชี้ j เพื่อปรับค่าและดันตัวตามหลังตัวหลักเทียมให้ไปอยู่ในตำแหน่งที่ถูกต้อง ซึ่งแสดงดังรูปที่ 3.3 หลังจบการแบ่งกั้นข้อมูล ตัวนำหน้าตัวหลักเทียมจะเป็นตัวนำหน้าตัวหลักและอยู่ติดกับตัวหลักทางด้านซ้าย ส่วนตัว

ตามหลังตัวหลักเทียมจะเป็นตัวตามหลังตัวหลักและอยู่ติดกับตัวหลักทางด้านขวา จากหลักการข้างต้นสามารถเขียนเป็นรหัสคำสั่งเทียมได้ดังนี้

รูปที่ 3.3 การแบ่งกั้นข้อมูลด้วยตัวนำหน้าและตัวตามหลังของตัวหลักที่ใช้การเรียงลำดับแบบฟอง

```
Function PSpartition_bubble (A[ ], left, mid, right)
    pivot = A[right];
    Swap A[mid] and A[right – 1];
    i = left + 1;
    j = right – 2;

    LOOP
        WHILE (A[i] < pivot) DO
            IF (A[i] < A[i – 1]) THEN
                Swap A[i] and A[i – 1];
            ENDIF
            i = i + 1;
        ENDWHILE
        WHILE (A[j] ≥ pivot) DO
            IF (A[j] > A[j + 1]) THEN
                Swap A[j] and A[j + 1];
            ENDIF
            j = j – 1;
        ENDWHILE
        IF (i ≥ j) THEN
            Exit LOOP;
        ENDIF
        Swap A[i] and A[j];
    ENDLOOP

    Move pivot to the correct position.
    Return index of PD and SC.
End PSpartition_bubble
```

ผู้วิจัยได้ทำการทดลองเพื่อทดสอบประสิทธิภาพของ PSQuicksort ด้วยการแบ่งกั้นข้อมูลทั้งสองแบบโดยเปรียบเทียบกับควิกซอร์ตที่มาจากงานวิจัยของ Hoare (HQuicksort) [2] การเรียงลำดับแบบผสาน (Mergesort) [20] ฮีปซอร์ต (Heapsort) [20] ซึ่งเป็นขั้นตอนวิธีการเรียงลำดับที่รู้จักกันโดยทั่วไปและมีเวลาในการทำงานเฉลี่ยเป็น $O(n \log n)$ และควิกซอร์ตที่มา

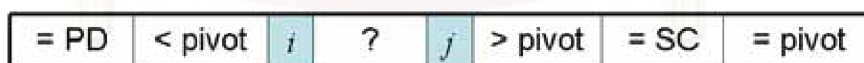
จากงานวิจัยของ Sedgewick (SWQuicksort) [5] โดยข้อมูลที่ใช้ทดสอบประสิทธิภาพเป็นข้อมูลแบบสุ่มซึ่งได้อธิบายรายละเอียดไว้ในบทที่ 4 ส่วนผลของการทดลองแสดงไว้ในภาคผนวก ข

จากการทดลองพบว่า PSQuicksort ด้วยการแบ่งกั้นข้อมูลด้วยตัวนำหน้าและตัวตามหลังของตัวหลักที่ใช้การค้นหาโดยตรงประมวลได้เร็วกว่าแบบที่การใช้การเรียงลำดับแบบฟอง แล้วยังประมวลผลได้เร็วกว่าการเรียงลำดับแบบผสาน ฮีปซอร์ต และ HQuicksort แต่ประมวลผลได้ช้ากว่า SWQuicksort เนื่องจาก PSQuicksort ใช้จำนวนครั้งในการเปรียบเทียบและการสลับที่มากกว่า SWQuicksort

ดังนั้นผู้วิจัยจึงทำการปรับปรุงขั้นตอนวิธีควิกซอร์ตด้วยตัวนำหน้าและตัวตามหลังของตัวหลัก โดยใช้ตัวนำหน้าตัวหลักและตัวตามหลังตัวหลักร่วมกับการแบ่งกั้นข้อมูลแบบสปลิตเอ็น (Split-end partition) [7] ซึ่งเป็นวิธีการแบ่งกั้นข้อมูลที่เหมาะสำหรับข้อมูลที่มีสมาชิกซ้ำกัน และเพิ่มข้อมูลลักษณะอื่นมาเพื่อทดสอบประสิทธิภาพของควิกซอร์ตด้วยตัวนำหน้าและตัวตามหลังของตัวหลักซึ่งมีรายละเอียดดังนี้

3.3 ขั้นตอนวิธีควิกซอร์ตด้วยตัวนำหน้าและตัวตามหลังของตัวหลักร่วมกับการแบ่งกั้นข้อมูลแบบสปลิตเอ็น

ควิกซอร์ตด้วยตัวนำหน้าตัวหลักและตัวตามหลังตัวหลักร่วมกับการแบ่งกั้นข้อมูลแบบสปลิตเอ็น (PSQsort) เป็นการนำการแบ่งกั้นแบบสปลิตเอ็น [7] มาใช้ร่วมกับควิกซอร์ตด้วยตัวนำหน้าและตัวตามหลังของตัวหลัก โดย PSQsort เพิ่มการเก็บสมาชิกที่มีค่าเท่ากับตัวนำหน้าตัวหลักและตัวตามหลังตัวหลักขณะที่ทำการแบ่งกั้นข้อมูล ซึ่งแสดงดังรูปที่ 3.4



รูปที่ 3.4 การแบ่งกั้นข้อมูลของ PSQsort

หลังจบขั้นตอนวิธีจึงทำการย้ายสมาชิกที่มีค่าเท่ากับตัวนำหน้าตัวหลัก ตัวหลักและตัวตามหลังตัวหลักไปอยู่ในตำแหน่งที่ถูกต้อง ทำให้ได้ข้อมูลย่อยสามกลุ่ม ได้แก่ กลุ่มข้อมูลย่อยที่สมาชิกมีค่าน้อยกว่าตัวนำหน้าตัวหลัก กลุ่มข้อมูลย่อยที่เรียงลำดับแล้วซึ่งเก็บสมาชิกที่มีค่าเท่ากับตัวนำหน้าตัวหลัก ตัวหลักและตัวตามหลังตัวหลัก และกลุ่มข้อมูลย่อยที่สมาชิกมีค่ามากกว่าตัวตามหลังตัวหลัก ซึ่งแสดงดังรูปที่ 3.5 จากหลักการข้างต้นสามารถเขียนเป็นรหัสคำสั่งเทียมได้ดังนี้

< PD	= PD, = pivot, = SC	> SC

รูปที่ 3.5 ลักษณะข้อมูลย่อยหลังการแบ่งกั้นข้อมูลแบบ PSQsort

```
Function PSQsort (A[ ], left, right)
    IF size of list A ≤ M THEN
        Call InsertionSort (A, left, right);
    ELSE
        Select pivot with Median of three;
        {ind_L, ind_R} = Split-end PSpartition(A, left, right);
        Recursively call PSQsort(A, left, ind_L);
        Recursively call PSQsort(A, ind_R, right);
    ENDIF
End PSQsort

Procedure Split-end PSpartition (A[ ], left, right)
    Set values of i, j, bound left (indL), bound right (indR), pivot, predecessor pivot
    (PD) and successor pivot (SC);

    LOOP
        WHILE (A[i] < pivot) DO
            IF (A[i] >= PD) THEN
                IF (A[i] == PD) THEN
                    Swap A[i] and A[indL];
                    indL = indL + 1;
                ELSE
                    Update value of PD and Swap A[i] and A[indL];
                    indL = indL + 1;
                ENDIF
            ENDIF
            i = i + 1;
        ENDWHILE
        WHILE (A[j] >= pivot) DO
            IF (A[j] == pivot) THEN
                Swap A[j] and A[indR];
                indR = indR – 1;
            ELSE If (A[j] >= SC) THEN
                IF (A[j] == SC) THEN
                    Swap A[j] and A[indR];
                    indR = indR – 1;
                ELSE
                    Update value of SC and Swap A[j] and A[indR];
                    indR = indR – 1;
                ENDIF
            ENDIF
```

ศูนย์วิทยทรัพยากร
จุฬาลงกรณ์มหาวิทยาลัย

```
        j = j – 1;
    ENDWHILE
    IF (i >= j) THEN
        Exit LOOP;
    Swap A[i] and A[j];
ENDLOOP

Move the elements are equal to pivot, PD and SC to the correct positions;
Return index of PD and SC;
End Split-end PSpartition
```

ผู้วิจัยได้ทำการทดลองเพื่อทดสอบประสิทธิภาพของ PSQsort โดยเปรียบเทียบกับควิกซอร์ตที่มาจากงานวิจัยของ Sedgewick (SWQuicksort) [5] และควิกซอร์ตที่มาจากงานวิจัยของ Bentley และ Mcilroy (qsort) [7] โดยข้อมูลที่ใช้ทดสอบประสิทธิภาพมีสี่แบบ ได้แก่ ข้อมูลแบบสุ่ม ข้อมูลที่เกือบเรียงลำดับ ข้อมูลที่เกือบเรียงลำดับแบบย้อนกลับ และข้อมูลแบบสุ่มที่มีสมาชิกซ้ำกัน โดยได้อธิบายรายละเอียดไว้ในบทที่ 4 ส่วนผลของการทดลองนี้แสดงไว้ในภาคผนวก ข

จากการทดลองพบว่า PSQsort ประมวลผลได้ช้ากว่า SWQuicksort และ qsort สำหรับข้อมูลแบบสุ่ม ข้อมูลที่เกือบเรียงลำดับ และข้อมูลที่เกือบเรียงลำดับแบบย้อนกลับ เนื่องจาก PSQsort ต้องเสียเวลาในการตรวจสอบสมาชิกที่มีค่าเท่ากับตัวนำหน้าตัวหลัก ตัวหลักและตัวตามหลังตัวหลักสำหรับกลุ่มข้อมูลที่ไม่ค่อยพบสมาชิกที่มีค่าเท่ากัน ส่วนข้อมูลแบบสุ่มที่มีสมาชิกซ้ำกันพบว่า PSQsort ประมวลผลได้เร็วกว่า SWQuicksort และ qsort เมื่อข้อมูลมีสมาชิกซ้ำกันมาก แต่ยังประมวลผลช้ากว่าเมื่อข้อมูลมีสมาชิกซ้ำกันน้อย เนื่องจากขั้นตอนวิธี PSQsort ต้องใช้จำนวนครั้งในการเปรียบเทียบและการสลับที่เป็นจำนวนมาก ถึงแม้ว่าจะสามารถลดจำนวนครั้งในการเรียกฟังก์ชันเวียนเกิดก็ตาม

ดังนั้นผู้วิจัยเสนอควิกซอร์ตด้วยตัวประชิดตัวหลักซึ่งเป็นการปรับปรุงขั้นตอนวิธี PSQsort โดยใช้เพียงตัวนำหน้าตัวหลักหรือตัวตามหลังตัวหลักร่วมกับตัวหลักสำหรับการแบ่งกั้นข้อมูล ซึ่งมีรายละเอียดดังนี้

ศูนย์วิทยทรัพยากร
จุฬาลงกรณ์มหาวิทยาลัย

3.4 ขั้นตอนวิธีควิกซอร์ตด้วยตัวประชิดตัวหลัก

ควิกซอร์ตด้วยตัวประชิดตัวหลัก (APQsort) เป็นขั้นตอนวิธีที่ปรับปรุงมาจาก PSQsort ซึ่ง APQsort มีหลักการทำงานคล้ายกับ PSQsort แต่ APQsort ใช้เพียงตัวนำหน้าตัวหลักหรือตัวตามหลังตัวหลักร่วมกับตัวหลักสำหรับการแบ่งกั้นข้อมูล เนื่องจากการใช้ทั้งตัวก่อนหน้าตัวหลัก ตัวหลักและตัวตามหลังตัวหลักสำหรับแบ่งกั้นข้อมูลมีขั้นตอนที่มีความซับซ้อนทำให้ต้องใช้จำนวนครั้งในการเปรียบเทียบและการสลับที่เป็นจำนวนมาก โดย APQsort จะเรียกใช้ขั้นตอนการแบ่งกั้นข้อมูลสามแบบ คือ การแบ่งกั้นข้อมูลด้วยตัวนำหน้าตัวหลัก (PD partition) การแบ่งกั้นข้อมูลด้วยตัวตามหลังตัวหลัก (SC partition) และการแบ่งกั้นข้อมูลแบบสปลิตเอ็น (Split-end partition) [7] ซึ่งการเรียกใช้การแบ่งกั้นข้อมูลทั้งสามแบบนั้นจะขึ้นอยู่กับค่าของตัวหลัก ตัวนำหน้าตัวหลักเทียม (p-PD) และตัวตามหลังตัวหลักเทียม (p-SC) ซึ่งมีรายละเอียดดังนี้

1) ตัวหลัก = ตัวนำหน้าตัวหลักเทียม = ตัวตามหลังตัวหลักเทียม ใช้การแบ่งกั้นข้อมูลแบบสปลิตเอ็น
2) ตัวหลักมีค่าใกล้กับตัวนำหน้าตัวหลักเทียมมากกว่าหรือเท่ากับตัวตามหลังตัวหลักเทียม
    a. ตัวหลัก = ตัวนำหน้าตัวหลักเทียม ใช้ SC partition
    b. ตัวหลัก $\neq$ ตัวนำหน้าตัวหลักเทียม ใช้ PD partition
3) ตัวหลักมีค่าใกล้กับตัวตามหลังตัวหลักเทียมมากกว่าตัวนำหน้าตัวหลักเทียม
    a. ตัวหลัก = ตัวตามหลังตัวหลักเทียม ใช้ PD partition
    b. ตัวหลัก $\neq$ ตัวตามหลังตัวหลักเทียม ใช้ SC partition

ขั้นตอนวิธีของการแบ่งกั้นข้อมูลด้วยตัวนำหน้าตัวหลัก และการแบ่งกั้นข้อมูลด้วยตัวตามหลังตัวหลักมีรายละเอียดดังนี้

ขั้นตอนวิธีการแบ่งกั้นข้อมูลด้วยตัวนำหน้าตัวหลัก

1) ย้ายสมาชิกที่มีค่าน้อยกว่าตัวหลักไปทางด้านซ้าย และถ้าสมาชิกนั้นมีค่ามากกว่าหรือเท่ากับตัวนำหน้าตัวหลักเทียม สมาชิกดังกล่าวจะถูกย้ายไปอยู่ทางขอบซ้ายของข้อมูล โดยปรับค่าของตัวนำหน้าตัวหลักเทียม ในกรณีที่สมาชิกตัวนั้นมีค่ามากกว่าตัวนำหน้าตัวหลักเทียม

2) ย้ายสมาชิกที่มีค่ามากกว่าหรือเท่ากับตัวหลักไปทางด้านขวา และถ้าสมาชิกนั้นมีค่าเท่ากับตัวหลักสมาชิกดังกล่าวจะถูกย้ายไปอยู่ทางขอบขวาของข้อมูล

หลังจบการแบ่งกั้นข้อมูล ค่าของตัวนำหน้าตัวหลักเทียมจะเท่ากับตัวนำหน้าตัวหลัก และข้อมูลจะมีลักษณะดังรูปที่ 3.6 จากนั้นย้ายสมาชิกที่มีค่าเท่ากับตัวหลักและตัวนำหน้าตัวหลักไปอยู่ในตำแหน่งที่ถูกต้อง ในกรณีที่ตำแหน่งของตัวนำหน้าตัวหลักอยู่ติดกับตำแหน่งของตัวหลักจะได้ว่าข้อมูลย่อยด้านซ้ายของตัวหลักเรียงลำดับแล้วและหยุดการทำซ้ำ

< PD	= PD	< pivot	> pivot	= pivot

รูปที่ 3.6 ลักษณะของข้อมูลหลังจบการแบ่งกั้นข้อมูลด้วยตัวนำหน้าตัวหลัก

ขั้นตอนวิธีการแบ่งกั้นข้อมูลด้วยตัวตามหลังตัวหลัก

1) ย้ายสมาชิกที่มีค่าน้อยกว่าหรือเท่ากับตัวหลักไปทางด้านซ้าย และถ้าสมาชิกนั้นมีค่าเท่ากับตัวหลักสมาชิกดังกล่าวจะถูกย้ายไปอยู่ทางขอบซ้ายของข้อมูล

2) ย้ายสมาชิกที่มีค่ามากกว่าตัวหลักไปทางด้านขวา และถ้าสมาชิกนั้นมีค่าน้อยกว่าหรือเท่ากับตัวตามหลังตัวหลักเทียม สมาชิกดังกล่าวจะถูกย้ายไปอยู่ทางขอบขวาของข้อมูล โดยปรับค่าของตัวตามหลังตัวหลักเทียม ในกรณีที่สมาชิกตัวนั้นมีค่าน้อยกว่าตัวตามหลังตัวหลักเทียม

หลังจบการแบ่งกั้นข้อมูล ค่าของตัวตามหลังตัวหลักเทียมจะเท่ากับตัวตามหลังตัวหลัก และข้อมูลจะมีลักษณะดังรูปที่ 3.7 จากนั้นย้ายสมาชิกที่มีค่าเท่ากับตัวหลักและตัวตามหลังตัวหลักไปอยู่ในตำแหน่งที่ถูกต้อง ในกรณีที่ตำแหน่งของตัวตามหลังตัวหลักอยู่ติดกับตำแหน่งของตัวหลักจะได้ว่าข้อมูลย่อยทางด้านขวาของตัวหลักเรียงลำดับแล้วและหยุดการทำซ้ำ

= pivot	< pivot	> pivot	= SC	> SC

รูปที่ 3.7 ลักษณะของข้อมูลหลังจบการแบ่งกั้นข้อมูลด้วยตัวตามหลังตัวหลัก

การแบ่งกั้นข้อมูลทั้งสามแบบจะได้ข้อมูลย่อยสามส่วนเหมือนกัน คือ ข้อมูลย่อยด้านซ้าย ข้อมูลย่อยที่เรียงลำดับแล้วและข้อมูลย่อยด้านขวา แต่ลักษณะสมาชิกข้อมูลย่อยของการแบ่งกั้นข้อมูลแต่ละแบบจะแตกต่างกันดังแสดงในตารางที่ 3.1 จากขั้นตอนข้างต้นสามารถเขียนเป็นรหัสคำสั่งเทียมได้ดังนี้

ศูนย์วิทยทรัพยากร
จุฬาลงกรณ์มหาวิทยาลัย

ข้อมูลย่อย	PD partition	SC partition	Split-end partition
ข้อมูลย่อยด้านซ้าย	สมาชิก < PD	สมาชิก < ตัวหลัก	สมาชิก < ตัวหลัก
ข้อมูลย่อยที่เรียงลำดับแล้ว	สมาชิก = PD และ สมาชิก = ตัวหลัก	สมาชิก = ตัวหลัก และสมาชิก = SC	สมาชิก = ตัวหลัก
ข้อมูลย่อยด้านขวา	สมาชิก > ตัวหลัก	สมาชิก > SC	สมาชิก > ตัวหลัก

ตารางที่ 3.1 ลักษณะสมาชิกในข้อมูลย่อยของการแบ่งกั้นข้อมูลสามแบบของ APQsort

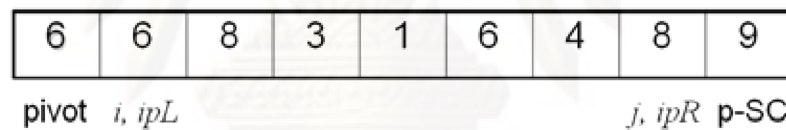
```
Function APQsort (A[ ], left, right)
    IF size of list A ≤ M THEN
        Call InsertionSort (A, left, right);
    ELSE
        Select pivot with Median of three;
        IF (pivot – p-PD == p-SC – pivot) THEN
            {ind_L, ind_R} = Split-end partition(A, left, right);
        ELSE IF (pivot – p-PD ≤ p-SC – pivot) THEN
            IF (pivot – p-PD ≠ 0) THEN
                {ind_L, ind_R} = PD partition(A, left, right);
            ELSE
                {ind_L, ind_R} = SC partition(A, left, right);
            ENDIF
        ElSE
            IF (p-SC – pivot ≠ 0) THEN
                {ind_L, ind_R} = SC partition(A, left, right);
            ELSE
                {ind_L, ind_R} = PD partition(A, left, right);
            ENDIF
        ENDIF

        Recursively call APQsort(A, left, ind_L);
        Recursively call APQsort(A, ind_R, right);
    ENDIF
End PSQsort

Function PD partition (A[ ], left, right)
    Set values of i, j, bound left (indL), bound right (indR), pivot and predecessor
    pivot (PD);

    LOOP
        WHILE (A[i] < pivot AND i ≤ j) DO
            IF (A[i] ≥ PD) THEN
                IF (A[i] == PD) THEN
                    Swap A[i] and A[indL];
                    indL =  indL + 1;
```

```
                ELSE
                    Update value of PD and Swap A[i] and A[indL];
                    indL = indL + 1;
                ENDIF
            ENDIF
            i = i + 1;
        ENDWHILE
        WHILE (A[j] >= pivot AND i ≤ j) DO
            IF (A[j] == pivot) THEN
                Swap A[j] and A[indR];
                indR = indR – 1;
            ENDIF
            j = j – 1;
        ENDWHILE
        IF (i ≥ j) THEN
            Exit LOOP;
        Swap A[i] and A[j];
    ENDLOOP

    Move the elements are equal to PD and pivot to the correct positions;
    Return index of PD and pivot;
End PD partition

Procedure SC partition (A[ ], left, right)
    Set values of i, j, bound left (indL), bound right (indR), pivot and successor pivot
    (SC);

    LOOP
        WHILE (A[i] <= pivot AND i ≤ j) DO
            IF (A[i] == pivot) THEN
                Swap A[i] and A[indL];
                indL = indL + 1;
            ENDIF
            i = i + 1;
        ENDWHILE
        WHILE (A[j] > pivot AND i ≤ j) DO
            IF (A[j] <= SC) THEN
                IF (A[j] == SC) THEN
                    Swap A[j] and A[indR];
                    indR = indR – 1;
                ELSE
                    Update value of SC and Swap A[j] and A[indR];
                    indR = indR – 1;
                ENDIF
            ENDIF
            j = j – 1;
        ENDWHILE
        IF (i ≥ j) THEN
```

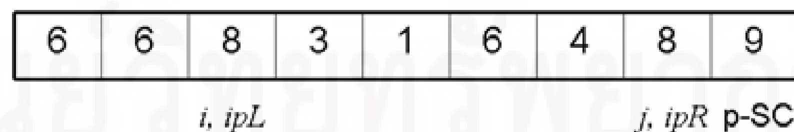
```
            Exit LOOP;
        Swap A[i] and A[j];
    ENDLOOP

    Move the elements are equal to pivot and SC to the correct positions;
    Return index of pivot and SC;
End SC partition
```

ตัวอย่างการทำงานของ APQsort โดยกำหนดข้อมูลดังนี้ 1, 6, 8, 3, 9, 6, 4, 8, 6

เริ่มต้นใช้มัธยฐานของสามตัวสำหรับกำหนดตัวหลัก พร้อมทั้งได้ตัวก่อนหน้าตัวหลักเทียมและตัวตามหลังตัวหลักเทียม โดยที่ตัวหลักมีค่าเท่ากับ 6 ตัวนำหน้าตัวหลักเทียมมีค่าเท่ากับ 1 และตัวตามหลังตัวหลักเทียมมีค่าเท่ากับ 9 ซึ่งพบว่าตัวตามหลังตัวหลักเทียมมีค่าใกล้ตัวหลักมากกว่าตัวนำหน้าตัวหลักเทียม ดังนั้นจึงใช้การแบ่งกั้นข้อมูลด้วยตัวตามหลังตัวหลัก ซึ่งตัวหลักจะอยู่ที่ขอบซ้ายส่วนตัวตามหลังตัวหลักจะอยู่ที่ขอบขวา และกำหนดให้ *i* กับ *ipL* เป็นตัวชี้จากตำแหน่งสอง และ *j* กับ *ipR* เป็นตัวชี้จากตำแหน่งรองสุดท้าย



ทำการเปรียบเทียบสมาชิกที่ตัวชี้ *i* กับตัวหลัก โดยขยับตัวชี้ *i* มาทางด้านขวาจนกระทั่งเจอสมาชิกที่มีค่ามากกว่า 6 พร้อมทั้งตรวจหาสมาชิกที่มีค่าเท่ากับตัวหลัก ซึ่งตรวจเจอสมาชิกตำแหน่งที่สองมีค่าเท่ากับตัวหลักจึงสลับที่สมาชิกตำแหน่งที่สองกับสมาชิกที่ตัวชี้ *ipL* ชี้อยู่ และขยับตัวชี้ *ipL* ไปทางขวาหนึ่งช่อง ตัวชี้ *i* หยุดที่สมาชิกตำแหน่งที่สามซึ่งมีค่าเท่ากับ 8



ทำการเปรียบเทียบสมาชิกที่ตัวชี้ *j* กับตัวหลัก โดยขยับตัวชี้ *j* มาทางด้านซ้ายจนกระทั่งเจอสมาชิกที่มีค่าน้อยกว่า 6 พร้อมทั้งตรวจหาสมาชิกที่มีค่ามากกว่าหรือเท่ากับตัวตามหลังตัวหลักเทียม ซึ่งตรวจเจอสมาชิกตำแหน่งที่แปดมีค่าน้อยกว่าตัวตามหลังตัวหลักเทียม จึงปรับค่าตัวตามหลังตัวหลักเทียมเป็น 8 พร้อมกับสลับที่สมาชิกตำแหน่งที่แปดกับสมาชิกที่ตัวชี้ *ipR* ชี้อยู่ และขยับตัวชี้ *ipR* มาทางซ้ายหนึ่งช่อง ตัวชี้ *j* หยุดที่สมาชิกตำแหน่งที่เจ็ดซึ่งมีค่าเท่ากับ 4

6	6	8	3	1	6	4	8	9

i, ipL *j, ipR* **p-SC**

ทำการเปรียบเทียบตัวชี้ *i* กับตัวชี้ *j* ซึ่งตัวชี้ *i* น้อยกว่า *j* จากนั้นทำการสลับสมาชิกระหว่างตัวชี้ *i* กับตัวชี้ *j*

6	6	4	3	1	6	8	8	9

i, ipL *j, ipR* **p-SC**

ทำการเปรียบเทียบสมาชิกที่ตัวชี้ *i* กับตัวหลัก โดยขยับตัวชี้ *i* มาทางด้านขวาจนกระทั่งเจอสมาชิกที่มีค่ามากกว่า 6 พร้อมทั้งตรวจหาสมาชิกที่มีค่าเท่ากับตัวหลัก ซึ่งตรวจเจอสมาชิกตำแหน่งที่หกมีค่าเท่ากับตัวหลักแล้วสลับที่สมาชิกตำแหน่งที่หกกับสมาชิกที่ตัวชี้ *ipL* ชี้อยู่ และขยับตัวชี้ *ipL* ไปทางขวาหนึ่งช่อง ตัวชี้ *i* หยุดที่สมาชิกตำแหน่งที่เจ็ดซึ่งมากกว่าหรือเท่ากับตัวชี้ *j* แล้วจบการแบ่งกั้นข้อมูลด้วยตัวตามหลังตัวหลัก

6	6	6	3	1	4	8	8	9

ipL *i, j, ipR* **p-SC**

หลังจบขั้นตอนวิธีได้ตัวตามหลังตัวหลักมีค่าเท่ากับ 8 จากนั้นทำการย้ายสมาชิกที่มีค่าเท่ากับตัวหลักและตัวตามหลังตัวหลักไปอยู่ ณ ตำแหน่งที่ถูกต้อง และได้กลุ่มข้อมูลย่อยสามกลุ่ม คือ กลุ่มข้อมูลย่อยที่สมาชิกมีค่าน้อยกว่าตัวหลัก กลุ่มข้อมูลย่อยที่เรียงลำดับแล้วซึ่งเก็บสมาชิกที่มีค่าเท่ากับตัวหลักกับตัวตามหลังตัวหลัก และกลุ่มข้อมูลย่อยที่สมาชิกมีค่ามากกว่าตัวตามหลังตัวหลัก จากนั้นทำการแบ่งข้อมูลด้วยวิธีการข้างต้นกับข้อมูลย่อยที่ยังไม่ได้เรียงลำดับจนกระทั่งไม่สามารถแบ่งข้อมูลได้อีกควิกซอร์ตก็จะหยุด ผลลัพธ์ที่ได้ก็คือข้อมูลที่เรียงลำดับจากน้อยไปมาก

4	1	3	<u>6</u>	<u>6</u>	<u>6</u>	<u>8</u>	<u>8</u>	9

ศูนย์วิทยทรัพยากร จุฬาลงกรณ์มหาวิทยาลัย

การวิเคราะห์เวลาการทำงานของควิกซอร์ตด้วยตัวประชิดตัวหลัก

การวิเคราะห์เวลาการทำงานของควิกซอร์ตด้วยตัวประชิดตัวหลักหาจากการแก้สมการเวียนเกิด จากขั้นตอนวิธีของควิกซอร์ตด้วยตัวประชิดตัวหลักที่ได้กล่าวมาในข้างต้น เวลาการทำงานของควิกซอร์ตด้วยตัวประชิดตัวหลักจะเท่ากับ เวลาการทำงานที่ควิกซอร์ตด้วยตัวประชิดตัวหลักใช้กับกลุ่มข้อมูลย่อยสองกลุ่ม บวกกับส่วนของการแบ่งกั้นข้อมูล ซึ่งสามารถเขียนเป็นสมการเวียนเกิดได้ดังนี้

$$\mathrm{T}(n) = \mathrm{T}(k) + \mathrm{T}(n-k-s) + cn$$

ซึ่ง T คือ เวลาการทำงานของควิกซอร์ตด้วยตัวประชิดตัวหลัก

n คือ จำนวนสมาชิกทั้งหมดในข้อมูล

k คือ จำนวนสมาชิกที่มีค่าน้อยตัวนำหน้าตัวหลัก

s คือ จำนวนสมาชิกของข้อมูลย่อยที่เรียงลำดับแล้ว

c คือ ค่าคงตัวใดๆ

โดยที่เวลาการทำงานของควิกซอร์ตด้วยตัวหลักประชิดสามารถแบ่งเป็นสามกรณีด้วยกันดังนี้

กรณีดีที่สุดเกิดขึ้นเมื่อจำนวนสมาชิกของข้อมูลย่อยที่เรียงลำดับแล้วเท่ากับจำนวนสมาชิกทั้งหมดในข้อมูล ทำให้จำนวนสมาชิกในข้อมูลย่อยทางซ้ายและทางขวาของตัวหลักเป็นศูนย์ ซึ่งเขียนสมการเวียนเกิดได้ดังนี้

$$\mathrm{T}(n) = 2\mathrm{T}(0) + cn$$

ซึ่ง $\mathrm{T}(0)$ ใช้เวลาการทำงานเป็นค่าคงที่ ดังนั้นจะได้

$$\mathrm{T}(n) = \mathrm{O}(n)$$

กรณีแย่ที่สุดเกิดขึ้นเมื่อตัวหลักเป็นค่ามากหรือน้อยที่สุดอันดับสองของข้อมูล และไม่มีสมาชิกอื่นที่มีค่าเท่ากับตัวนำหน้าตัวหลัก ตัวหลักและตัวตามหลังตัวหลัก ซึ่งทำให้รู้ตำแหน่งที่ถูกต้องของสมาชิกครั้งละสองตัวสำหรับทุกๆ รอบของการแบ่งกั้นข้อมูล และการเรียกฟังก์ชันเกิดจะหยุดเมื่อข้อมูลย่อยมีจำนวนสมาชิกน้อยกว่าหรือเท่ากับ 10 ตัว โดยใช้การเรียงลำดับแบบแทรกเรียงลำดับข้อมูลย่อยแทน APQsort ซึ่งแสดงดังรูปที่ 3.8 และสามารถเขียนเป็นสมการเวียนเกิดได้ดังนี้

$$\mathrm{T}(n) \quad = \mathrm{T}(0) + \mathrm{T}(n-2) + cn$$

ซึ่ง $\mathrm{T}(0)$ เป็นเวลาการทำงานของควิกซอร์ตที่ไม่มีสมาชิก ดังนั้น $\mathrm{T}(0)=0$ และจะได้

$$\mathrm{T}(n) \quad = \mathrm{T}(n-2) + cn$$

$$\begin{aligned} \mathrm{T}(n-2) &= \mathrm{T}(n-4) + c(n-2) \\ \mathrm{T}(n-4) &= \mathrm{T}(n-6) + c(n-4) \\ &\vdots \\ \mathrm{T}(12) &= \mathrm{T}(10) + c(12) \end{aligned}$$

เขียน $\mathrm{T}(n)$ ให้อยู่ในรูปของ n ได้ดังนี้

$$\mathrm{T}(n) \quad = \mathrm{T}(10) + c\left[10 + 12 + \ldots + (n-2) + n\right]$$

$$\mathrm{T}(n) \quad = \mathrm{T}(10) + c\left(\frac{n}{4}(n+2) - 20\right)$$

โดย $\mathrm{T}(10)$ เป็นเวลาของการเรียงลำดับแบบแทรกซึ่งประมาณเป็นค่าคงที่ดังนั้น

$$\mathrm{T}(n) \quad = \mathrm{O}\left(n^2\right)$$

ศูนย์วิทยทรัพยากร
จุฬาลงกรณ์มหาวิทยาลัย

รูปที่ 3.8 การทำงานของควิกซอร์ตด้วยตัวประชิดตัวหลักในกรณีแย่ที่สุด

ศูนย์วิทยทรัพยากร
จุฬาลงกรณ์มหาวิทยาลัย

บทที่ 4

การทดลองและผลการทดลอง

จากแนวคิดเบื้องต้นในบทที่ 2 จนถึงขั้นตอนวิธีปรับปรุงควิกซอร์ตในบทที่ 3 ผู้วิจัยจะกล่าวถึงการเตรียมข้อมูล การทดสอบประสิทธิภาพของขั้นตอนวิธีและการเปรียบเทียบผลการทดลอง พร้อมสรุป ซึ่งมีรายละเอียดดังต่อไปนี้

4.1 การทดลอง

4.1.1 เครื่องมือและภาษาโปรแกรมที่ใช้ในการทดลอง

โปรแกรมรหัสคำสั่งทั้งหมดที่ใช้ในงานวิจัยนี้เขียนด้วยภาษาซีโดยใช้โปรแกรม Dev C++ รุ่น 4.9.9.2 ประมวลผลโดยใช้คอมพิวเตอร์ซีพียูอินเทลคอร์ทูดูอัลที่มีความเร็ว 2.0 GHz พร้อมหน่วยความจำหลัก 2 GB ระบบปฏิบัติการวินโดว์วิสต้า

4.1.2 ข้อมูลที่ใช้ทดลอง

ข้อมูลที่ใช้ทดลองเป็นแบบจำนวนเต็มเท่านั้น โดยที่ข้อมูลจะมีค่าอยู่ระหว่าง 1 ถึง n ซึ่ง n แทนขนาดของข้อมูลแต่ละชุด โดยแบ่งการจำลองข้อมูลออกเป็นสี่ลักษณะดังนี้

1) ข้อมูลแบบที่หนึ่ง เป็นข้อมูลสุ่มที่มีการแจกแจงแบบเอกรูป (Uniform Distribution) ข้อมูลแบบนี้เป็นที่นิยมใช้กันโดยทั่วไปสำหรับทดสอบประสิทธิภาพในการทำงานของขั้นตอนวิธีการเรียงลำดับ โดยมีด้วยกันสามขนาด คือ 10000 30000 และ 100000 อย่างละ 100 แฟ้มข้อมูล

2) ข้อมูลแบบที่สอง เป็นข้อมูลสุ่มที่เกือบเรียงลำดับ [16] ข้อมูลชนิดนี้เป็นที่นิยมนำมาใช้สำหรับทดสอบประสิทธิภาพการทำงานของควิกซอร์ตเนื่องจากเป็นข้อมูลที่ทำให้เกิดกรณีแย่ที่สุดของควิกซอร์ต ข้อมูลแบบนี้มีขนาดเท่ากับ 30000 โดยมีอัตราส่วนที่ไม่เรียงลำดับเป็น 0 0.01 0.03 0.1 และ 0.3 อย่างละ 100 แฟ้มข้อมูล โดยที่อัตราส่วนที่ไม่เรียงลำดับคำนวณได้จากจำนวนสมาชิกที่ไม่เรียงลำดับหารด้วยจำนวนข้อมูลทั้งหมด
3) ข้อมูลแบบที่สาม เป็นข้อมูลสุ่มที่เกือบเรียงลำดับแบบย้อนกลับ [16] ข้อมูลชนิดนี้เป็นข้อมูลที่นิยมนำมาใช้สำหรับทดสอบประสิทธิภาพการทำงานของควิกซอร์ตเนื่องจากเป็นข้อมูลที่ทำให้เกิดกรณีแย่ที่สุดของควิกซอร์ต ข้อมูลแบบนี้มีขนาดเท่ากับ 30000 โดยมีอัตราส่วนที่ไม่เรียงลำดับแบบย้อนกลับเป็น 0 0.01 0.03 0.1 และ 0.3 อย่างละ 100 แฟ้มข้อมูล โดยที่อัตราส่วนที่ไม่เรียงลำดับแบบย้อนกลับคำนวณได้จาก จำนวนสมาชิกที่ไม่เรียงลำดับแบบย้อนกลับหารด้วยจำนวนข้อมูลทั้งหมด
4) ข้อมูลแบบที่สี่ เป็นชุดข้อมูลสุ่มที่มีการแจกแจงแบบเอกรูปที่มีสมาชิกซ้ำกัน [19] ข้อมูลชนิดนี้เป็นข้อมูลที่งานวิจัยนี้สนใจ ข้อมูลแบบนี้มีขนาดเท่ากับ 30000 โดยมีความไม่ซ้ำกันของสมาชิกเป็น 0.001 0.01 0.03 0.1 และ 0.3 อย่างละ 100 แฟ้มข้อมูล โดยที่ความไม่ซ้ำกันของสมาชิก d หมายถึง ข้อมูลจะมีค่าระหว่าง 1 ถึง dN

4.2 ผลการทดลอง

งานวิจัยนี้เสนอควิกซอร์ตด้วยตัวประชิดตัวหลัก (APQsort) เป็นวิธีการปรับปรุงควิกซอร์ตที่เน้นการลดจำนวนครั้งที่ใช้ฟังก์ชันเวียนเกิด โดยใช้ตัวก่อนหน้าตัวหลักหรือตัวตามหลังตัวหลักร่วมกับการแบ่งกั้นข้อมูลแบบสปลิตเอ็นเพื่อเร่งเวลาในการประมวลสำหรับข้อมูลที่มีสมาชิกซ้ำกัน ดังนั้นผู้วิจัยจึงใช้เวลาประมวลผลเป็นตัวตัดสินประสิทธิภาพของขั้นตอนวิธี ซึ่งเวลาที่ใช้ในการประมวลผลเป็นเวลาเฉลี่ยจากการทดลองซ้ำ 100 ครั้งกับข้อมูลทั้งหมด โดยใช้คำสั่ง clock ในภาษาซีช่วยนับเวลาซึ่งมีหน่วยเป็นวินาที และใช้ paired t-test [21] สำหรับการทดสอบสถิติเพื่อบ่งบอกความแตกต่างอย่างมีนัยสำคัญของเวลาการทำงานเฉลี่ยโดยใช้ $\alpha = 0.05$ รวมถึงแสดงจำนวนครั้งที่ใช้การดำเนินการต่างๆ ได้แก่ การสลับที่ การเปรียบเทียบ การเรียกฟังก์ชันเวียนเกิด

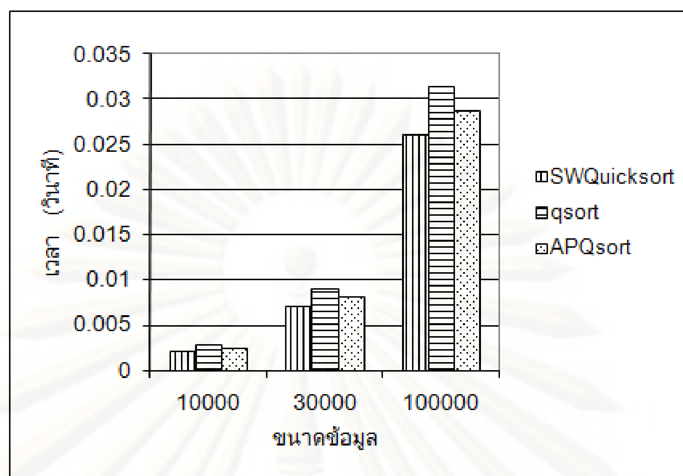
ขั้นตอนวิธีที่นำมาใช้เปรียบเทียบ APQsort ได้แก่

1) qsort [7] เป็นขั้นตอนวิธีที่ปรับปรุงมาจากควิกซอร์ตที่เสนอโดย Bentley กับ Mcilroy ซึ่งใช้การแบ่งกั้นข้อมูลแบบสปลิตเอ็น ใช้มัธยฐานของสามตัวและมัธยฐานของเก้าตัวสำหรับกำหนดตัวหลัก และใช้การเรียงลำดับแบบแทรกเมื่อข้อมูลย่อยมีขนาดเล็ก

2) SWQuicksort [5] เป็นขั้นตอนวิธีที่ปรับปรุงมาจากควิกซอร์ตที่เสนอโดย Sedgewick ซึ่งใช้มัธยฐานของสามกำหนดตัวหลัก ควิกซอร์ตที่ไม่มีการเรียกฟังก์ชันเวียนเกิด และใช้การเรียงลำดับแบบแทรกเมื่อข้อมูลย่อยมีขนาดเล็ก

การทดลองกับข้อมูลแบบที่หนึ่งพบว่า SWQuicksort ประมวลผลเฉลี่ยเร็วที่สุด ส่วน APQsort ประมวลผลเฉลี่ยเร็วกว่า qsort ซึ่งแสดงดังรูปที่ 4.1 เนื่องจาก APQsort ต้องเสียเวลาในการตรวจหาสมาชิกที่มีค่าเท่ากับตัวหลักและตัวนำหน้าตัวหลักหรือตัวตามหลังตัวหลัก และ qsort ต้องเสียเวลาในการตรวจหาสมาชิกที่มีค่าเท่ากับตัวหลัก แต่ข้อมูลที่ทำการทดลองนั้นมักไม่พบสมาชิกที่มีค่าเท่ากัน ดังนั้น APQsort และ qsort จึงประมวลผลช้ากว่า SWQuicksort ส่วนที่ APQsort ประมวลผลเร็วกว่า qsort เนื่องจาก APQsort มีจำนวนครั้งที่เรียกฟังก์ชันเวียนเกิดน้อยกว่า qsort มาก ซึ่งการดำเนินการต่างๆ ของขั้นตอนวิธีทั้งสามแสดงดังตารางที่ 4.1

ศูนย์วิทยทรัพยากร
จุฬาลงกรณ์มหาวิทยาลัย



รูปที่ 4.1 การเปรียบเทียบเวลาการทำงานของข้อมูลแบบสุ่ม

Sizes	Methods	Compare	Swap	Recursive
10000	SWQuicksort	177,677	38,252	0
	qsort	332,400	34,049	8,342
	APQsort	327,783	40,373	1,155
30000	SWQuicksort	599,787	125,883	0
	qsort	1,131,770	113,927	24,707
	APQsort	1,124,830	132,428	3,451
100000	SWQuicksort	2,240,760	426,412	0
	qsort	4,046,070	435,956	42,426
	APQsort	4,093,530	453,174	8,956

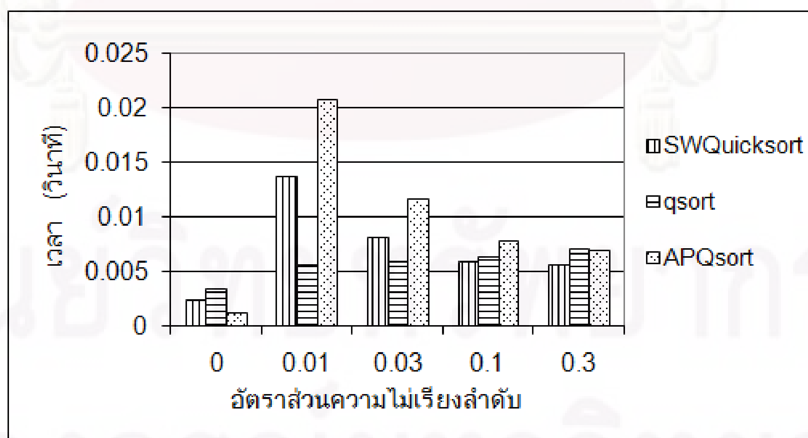
ตารางที่ 4.1 การดำเนินการทั้งสามอย่างสำหรับข้อมูลแบบสุ่ม

การทดลองกับข้อมูลแบบที่สองพบว่า เวลาในการประมวลผลของขั้นตอนวิธีทั้งสามสำหรับข้อมูลที่เกือบเรียงลำดับจะขึ้นอยู่กับอัตราส่วนความไม่เรียงลำดับซึ่งแสดงดังรูปที่ 4.2 โดยแบ่งข้อมูลออกเป็นสามกรณีดังแสดงในตารางที่ 4.2

1) อัตราส่วนที่ไม่เรียงลำดับเป็นศูนย์ คือ ข้อมูลมีลักษณะเรียงลำดับ พบว่า APQsort ประมวลผลเฉลี่ยเร็วที่สุดอย่างมีนัยสำคัญ เนื่องจาก APQsort สามารถตรวจสอบการเรียงลำดับของข้อมูลย่อยได้ ทำให้รู้ตำแหน่งที่ถูกต้องของสมาชิกทีละครึ่งหนึ่งสำหรับทุกๆ รอบของการแบ่งกั้นข้อมูล

2) อัตราส่วนที่ไม่เรียงลำดับมีค่าใกล้ศูนย์ ได้แก่ อัตราส่วนที่ไม่เรียงลำดับเป็น 0.01 กับ 0.03 คือ ข้อมูลมีลักษณะเกือบเรียงลำดับ พบว่า qsort ประมวลผลเฉลี่ยเร็วที่สุด เนื่องจาก SWQuicksort กับ APQsort ใช้การเปรียบเทียบเป็นจำนวนมาก สำหรับ APQsort ที่ใช้จำนวนการเปรียบเทียบและการสลับที่เพราะต้องทำการปรับค่าตัวนำหน้าตัวหลักเทียมหรือตัวตามหลังตัวหลักเทียมบ่อยครั้ง

3) อัตราส่วนที่ไม่เรียงลำดับมีค่าใกล้เคียงกับข้อมูลแบบสุ่ม ได้แก่ อัตราส่วนที่ไม่เรียงลำดับเป็น 0.1 กับ 0.3 พบว่า SWQuicksort ประมวลผลเฉลี่ยเร็วที่สุด เนื่องจากไม่เสียเวลาในการเรียกฟังก์ชันเวียนเกิดและใช้การเปรียบเทียบเป็นจำนวนน้อยกว่าขั้นตอนวิธีอื่น



รูปที่ 4.2 การเปรียบเทียบเวลาการทำงานของข้อมูลที่เกือบเรียงลำดับ

Unsorted Ratio	Method	Compare	Swap	Recursive
0	SWQuicksort	422,773	8,190	0
	qsort	796,086	8,190	8,191
	APQsort	150,059	29,993	12
0.01	SWQuicksort	3,055,370	58,956	0
	qsort	966,789	43,767	33,280
	APQsort	6,317,630	72,240	6,125
0.03	SWQuicksort	1,615,520	64,819	0
	qsort	978,725	50,210	32,673
	APQsort	3,183,500	76,395	5,650
0.1	SWQuicksort	954,877	74,637	0
	qsort	992,949	57,974	31,608
	APQsort	1,744,160	84,389	4,634
0.3	SWQuicksort	744,410	89,333	0
	qsort	1,025,060	72,107	29,680
	APQsort	1,254,520	97,737	3,838

ตารางที่ 4.2 การดำเนินการทั้งสามอย่างสำหรับข้อมูลที่เกือบเรียงลำดับ

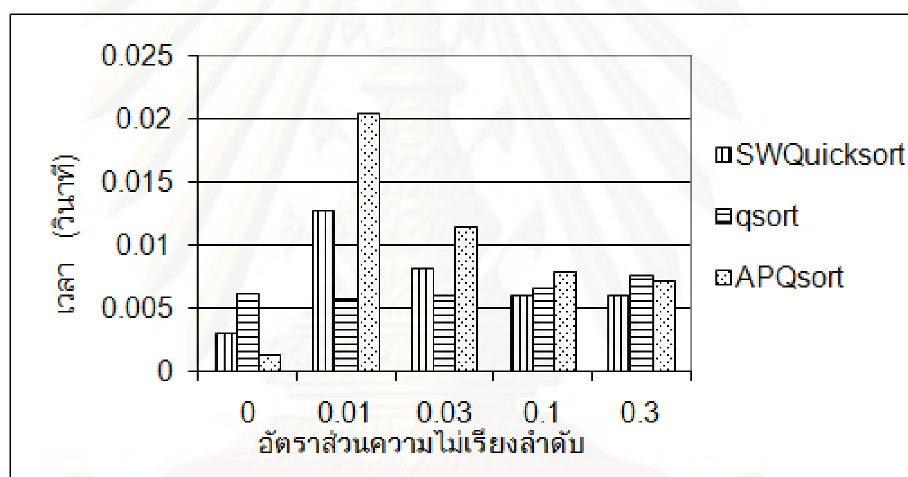
การทดลองกับข้อมูลแบบที่สามพบว่า เวลาในการประมวลผลของขั้นตอนวิธีทั้งสามสำหรับข้อมูลที่เกือบเรียงลำดับแบบย้อนกลับจะขึ้นอยู่กับอัตราส่วนความไม่เรียงลำดับซึ่งแสดงดังรูปที่ 4.3 โดยแบ่งข้อมูลออกเป็นสามกรณีดังนี้ ซึ่งการดำเนินการต่างๆ ของขั้นตอนวิธีทั้งสามแสดงดังตารางที่ 4.3

1) อัตราส่วนที่ไม่เรียงลำดับเป็นศูนย์ คือ ข้อมูลมีลักษณะเรียงลำดับแบบย้อนกลับ พบว่า APQsort ประมวลผลเฉลี่ยเร็วที่สุดอย่างมีนัยสำคัญ เนื่องจาก APQsort สามารถตรวจสอบการเรียงลำดับของข้อมูลย่อยได้ ทำให้รู้ตำแหน่งที่ถูกต้องของสมาชิกทีละครึ่งนึงสำหรับทุกๆ รอบของการแบ่งกั้นข้อมูล

2) อัตราส่วนที่ไม่เรียงลำดับมีค่าใกล้ศูนย์ ได้แก่ อัตราส่วนที่ไม่เรียงลำดับเป็น 0.01 กับ 0.03 คือ ข้อมูลมีลักษณะเกือบเรียงลำดับแบบย้อนกลับ พบว่า qsort ประมวลผลเฉลี่ยเร็วที่สุด เนื่องจาก SWQuicksort กับ APQsort ใช้การเปรียบเทียบเป็นจำนวนมาก สำหรับ APQsort

ที่ใช้จำนวนการเปรียบเทียบและการสลับที่เพราะต้องทำการปรับค่าตัวนำหน้าตัวหลักเทียมหรือตัวตามหลังตัวหลักเทียมบ่อยครั้ง

3) อัตราส่วนที่ไม่เรียงลำดับมีค่าใกล้เคียงกับข้อมูลแบบสุ่ม ได้แก่ อัตราส่วนที่ไม่เรียงลำดับเป็น 0.1 กับ 0.3 พบว่า SWQuicksort ประมวลผลเฉลี่ยเร็วที่สุด เนื่องจากไม่เสียเวลาในการเรียกฟังก์ชันเวียนเกิดและใช้การเปรียบเทียบเป็นจำนวนน้อยกว่าขั้นตอนวิธีอื่น



รูปที่ 4.3 การเปรียบเทียบเวลาการทำงานของข้อมูลเกือบเรียงลำดับแบบย้อนกลับ

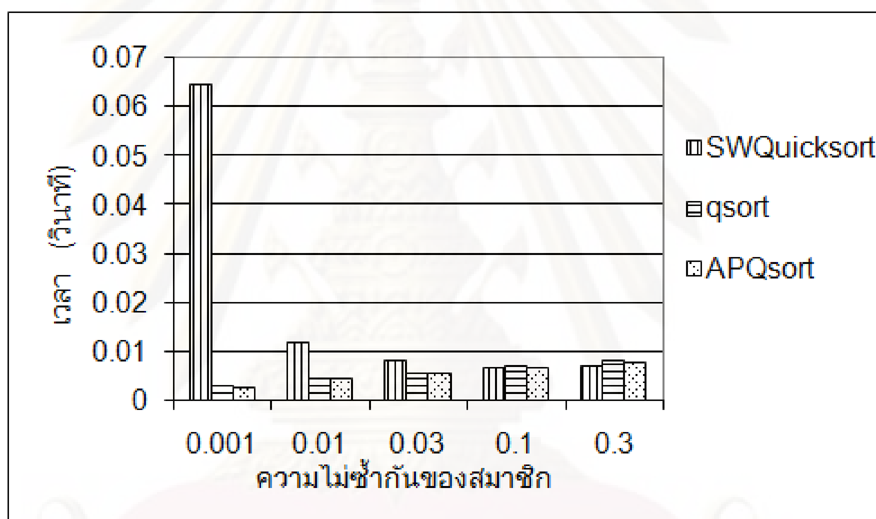
Unsorted Ratio	Method	Compare	Swap	Recursive
0	SWQuicksort	467,770	23,190	0
	qsort	1,173,180	60,394	20,377
	APQsort	195,053	44,997	12
0.01	SWQuicksort	2,785,270	70,403	0
	qsort	1,009,750	56,329	33,011
	APQsort	6,189,330	86,973	6,209
0.03	SWQuicksort	1,579,790	77,014	0
	qsort	1,018,690	62,752	32,757
	APQsort	3,056,180	89,780	5,468
0.1	SWQuicksort	948,546	88,210	0
	qsort	1,042,550	73,941	31,615
	APQsort	1,753,310	98,354	4,573
0.3	SWQuicksort	767,823	105,615	0
	qsort	1,088,980	92,087	29,803
	APQsort	1,249,700	113,249	3,829

ตารางที่ 4.3 การดำเนินการทั้งสามอย่างสำหรับข้อมูลที่เกือบเรียงลำดับแบบย้อนกลับ

การทดลองกับข้อมูลแบบที่สี่พบว่า เวลาในการประมวลผลของขั้นตอนวิธีทั้งสามสำหรับข้อมูลแบบสุ่มที่มีสมาชิกซ้ำกันจะขึ้นอยู่กับความไม่ซ้ำกันของสมาชิกซึ่งแสดงดังรูปที่ 4.4 โดยแบ่งข้อมูลออกเป็นสองกรณีดังนี้ ซึ่งการดำเนินการต่างๆ ของขั้นตอนวิธีทั้งสามแสดงดังตารางที่ 4.4

1) ความไม่ซ้ำกันของสมาชิกมีค่าใกล้ศูนย์ ได้แก่ ความไม่ซ้ำกันของสมาชิกเป็น 0.001 0.01 และ 0.03 คือ ข้อมูลมีจำนวนสมาชิกที่มีค่าเท่ากันเป็นจำนวนมาก พบว่า APQsort ประมวลผลเฉลี่ยเร็วที่สุดอย่างมีนัยสำคัญ เนื่องจาก APQsort ใช้การดำเนินการโดยรวมน้อยกว่าขั้นตอนวิธีอื่น ส่วน SWQuicksort ใช้การเปรียบเทียบเป็นจำนวนมาก เนื่องจากการแบ่งกั้นข้อมูลของ SWQuicksort จะแบ่งข้อมูลออกเป็น ข้อมูลย่อยที่สมาชิกมีค่าน้อยกว่าตัวหลักกับข้อมูลย่อยที่สมาชิกมีค่ามากกว่าหรือเท่ากับตัวหลัก ซึ่งถ้าข้อมูลมีสมาชิกซ้ำกันมากๆ จะทำให้การแบ่งกั้นข้อมูลมีลักษณะใกล้เคียงกับกรณีแย่ที่สุดของควิกซอร์ต

2) อัตราส่วนที่ไม่เรียงลำดับมีค่าใกล้เคียงกับข้อมูลแบบสุ่ม ได้แก่ ความไม่ซ้ำกันของสมาชิกเป็น 0.1 กับ 0.3 คือ ข้อมูลมีจำนวนสมาชิกที่มีค่าเท่ากันเป็นจำนวนน้อย พบว่า SWQuicksort ประมวลผลเฉลี่ยเร็วที่สุด เนื่องจาก SWQuicksort ใช้การดำเนินการโดยรวมน้อยกว่าขั้นตอนวิธีอื่น ส่วน APQsort ประมวลผลเฉลี่ยเร็วกว่า qsort เนื่องจาก APQsort ใช้การเปรียบเทียบและการสลับที่ใกล้เคียงกับ qsort แต่สามารถลดการเรียกฟังก์ชันเวียนเกิดได้เป็นจำนวนมาก



รูปที่ 4.4 การเปรียบเทียบเวลาการทำงานของข้อมูลสุ่มที่มีสมาชิกซ้ำกัน

Unrepeated	Method	Compare	Swap	Recursive
0.001	SWQuicksort	15,571,100	65,033	0
	qsort	349,170	68,798	30
	ASQsort	342,561	68,951	18
0.01	SWQuicksort	2,221,710	86,156	0
	qsort	626,895	92,811	300
	ASQsort	624,832	91,252	179
0.03	SWQuicksort	1,225,800	91,702	0
	qsort	768,823	105,892	900
	ASQsort	769,560	102,474	535
0.1	SWQuicksort	788,523	91,901	0
	qsort	934,267	119,524	3,003
	ASQsort	930,702	112,784	1,456
0.3	SWQuicksort	762,706	113,884	0
	qsort	1,049,220	118,683	11,440
	ASQsort	1,055,550	122,536	2,582

ตารางที่ 4.4 การดำเนินการทั้งสามอย่างสำหรับข้อมูลแบบสุ่มที่มีสมาชิกซ้ำกัน

ศูนย์วิทยทรัพยากร
จุฬาลงกรณ์มหาวิทยาลัย

บทที่ 5

สรุปผลการทดลอง

งานวิจัยนี้เสนอควิกซอร์ตด้วยตัวประชิดตัวหลัก (APQsort) เป็นการปรับปรุงควิกซอร์ตให้มีประสิทธิภาพดีขึ้นสำหรับข้อมูลที่มีสมาชิกซ้ำกัน โดยเน้นการลดจำนวนครั้งที่เรียกฟังก์ชันเวียนเกิดซึ่งใช้ประโยชน์จากตัวนำหน้าตัวหลักหรือตัวตามหลังตัวหลักร่วมกับการแบ่งกั้นข้อมูลแบบสปลิตเอ็น APQsort เพิ่มการเก็บสมาชิกที่มีค่าเท่ากับตัวนำหน้าตัวหลักหรือตัวตามหลังตัวหลักทำให้รู้ตำแหน่งที่ถูกต้องของสมาชิกเพิ่มขึ้นจากวิธีการแบ่งกั้นข้อมูลแบบสปลิตเอ็น และช่วยลดจำนวนการเรียกฟังก์ชันเวียนเกิดลง

จากการทดลองพบว่า เวลาการประมวลผลของ APQsort ขึ้นอยู่กับลักษณะของข้อมูล ซึ่งแบ่งเป็น 4 ลักษณะดังนี้

1) ข้อมูลสุ่มและข้อมูลสุ่มที่มีสมาชิกซ้ำกันเป็นจำนวนน้อย SWQuicksort ประมวลผลเฉลี่ยเร็วกว่า APQsort เนื่องจาก การตรวจหาสมาชิกที่มีค่าเท่ากันต้องเพิ่มจำนวนการเปรียบเทียบและการสลับที่ แต่ข้อมูลที่ทำการทดลองนั้นมักไม่พบสมาชิกที่มีค่าเท่ากันหรือพบเป็นจำนวนน้อย และ SWQuicksort เป็นขั้นตอนวิธีที่ไม่ต้องเสียเวลาในการเรียกฟังก์ชันเวียนเกิด ส่วน APQsort ประมวลผลเฉลี่ยได้เร็วกว่า qsort เนื่องจาก ในแต่ละรอบของการแบ่งกั้นข้อมูล APQsort รู้ตำแหน่งที่ถูกต้องของสมาชิกมากกว่า qsort ทำให้มีจำนวนครั้งที่เรียกฟังก์ชันเวียนเกิดน้อยกว่า

2) ข้อมูลเรียงลำดับและข้อมูลเรียงลำดับแบบย้อนกลับ APQsort ประมวลผลเฉลี่ยเร็วกว่า SWQuicksort และ qsort เนื่องจาก APQsort สามารถตรวจสอบการเรียงลำดับของข้อมูลย่อยได้ ทำให้รู้ตำแหน่งที่ถูกต้องของสมาชิกทีละครึ่งนึงสำหรับทุกๆ รอบของการแบ่งกั้นข้อมูลซึ่งช่วยลดจำนวนการเปรียบเทียบและการเรียกฟังก์ชันเวียนเกิด

3) ข้อมูลเกือบเรียงลำดับและข้อมูลเกือบเรียงลำดับแบบย้อนกลับ APQsort ประมวลผลเฉลี่ยช้ากว่า SWQuicksort และ qsort เนื่องจาก APQsort เสียเวลาในการปรับค่าตัวนำหน้าตัวหลักเทียมหรือตัวตามหลังตัวหลักเทียมบ่อยครั้ง ซึ่งต้องใช้การเปรียบเทียบและการสลับที่เป็นจำนวนมาก

4) ข้อมูลสุ่มที่มีสมาชิกซ้ำกันมาก APQsort ประมวลผลเฉลี่ยเร็วกว่า SWQuicksort และ qsort ถึงแม้ว่าการตรวจหาสมาชิกที่มีค่าเท่ากับตัวนำหน้าตัวหลักหรือตัวตามหลังตัวหลักของ APQsort ต้องเพิ่มการเปรียบเทียบและการสลับที่เป็นจำนวนมาก แต่ทำให้รู้ตำแหน่งที่ถูกต้องของสมาชิกและช่วยลดการดำเนินการต่างๆ ลงได้เป็นจำนวนมากเช่นเดียวกัน ดังนั้น APQsort จึงประมวลผลได้เร็วกว่า qsort

จากผลการทดลองจึงสรุปข้อดีและข้อเสียของ APQsort ได้ดังนี้

- ข้อดี คือ APQsort สามารถรู้ตำแหน่งที่ถูกต้องของสมาชิกเป็นจำนวนมากถ้าข้อมูลมีสมาชิกซ้ำกันมาก และ APQsort สามารถตรวจสอบการเรียงลำดับของข้อมูลย่อยได้ ซึ่งทำให้ประมวลผลได้เร็วสำหรับข้อมูลที่เรียงลำดับหรือเรียงลำดับแบบย้อนกลับ และข้อมูลที่มีสมาชิกซ้ำกันมาก เช่น ข้อมูลอายุ ข้อมูลวันที่ ข้อมูลเดือน ข้อมูลปี ข้อมูลน้ำหนัก ข้อมูลส่วนสูง เป็นต้น

- ข้อเสีย คือ การหาตัวนำหน้าตัวหลักหรือตัวตามหลังตัวหลัก และการหาสมาชิกที่มีค่าเท่ากันต้องเพิ่มจำนวนการเปรียบเทียบและการสลับที่ ถ้าข้อมูลไม่มีสมาชิกที่มีค่าซ้ำกันหรือข้อมูลมีสมาชิกที่มีค่าซ้ำกันแต่มีจำนวนน้อยแล้ว ทำให้การหาสมาชิกที่มีค่าเท่ากันเป็นค่าใช้จ่ายที่ไม่ทำให้เกิดประโยชน์ และถ้าข้อมูลเป็นแบบเกือบเรียงลำดับหรือข้อมูลเป็นแบบเกือบเรียงลำดับแบบย้อนกลับแล้ว ทำให้ต้องปรับค่าของตัวนำหน้าตัวหลักหรือตัวตามหลังตัวหลักบ่อยครั้ง ซึ่งต้องใช้การเปรียบเทียบและการสลับที่เป็นจำนวนมาก

รายการอ้างอิง

[1] Cormen, T.H., Leiserson C.E., Rivest, R.L. and Stein, C. 2001. INTRODUCTION TO ALGORITHMS. Cambridge, MA : MIT Press.

[2] Hoare, C.A.R. 1962. Algorithm 63 Partition and Algorithm 64 Quicksort. Communications of the ACM 4(7) : 321.

[3] Scowen, R.S. 1965. Algorithm 271 Quickersort, Communications of the ACM 8(11) : 669 – 670.

[4] Singleton, R.C. 1969. Algorithm 347 An efficient algorithm for sorting with minimal storage. Communications of ACM 12(3) : 186 – 187.

[5] Sedgewick, R. 1978. Implementing quicksort programs. Communications of the ACM 21(10) : 847 – 857.

[6] Wainwright, R.L. 1985. A class of sorting algorithms based on quicksort. Communication of the ACM : 28(4), 396 – 402.

[7] Bentley, J.L. and Mcilroy, M.D. 1993. Engineering a sort function. Software – Practice and Experience 23(11) : 1249 – 1265.

[8] Knuth, D.E. 1973. The art of Computer Programming Vol. 3: Sorting and Searching. CA : Addison-Wesley Press.

[9] Astrachan, O. 2003. Bubble sort: An archaeological algorithmic analysis. ACM SIGCSE 35(1) : 1 – 5.

[10] Shell, D.L. 1959. A high-speed sorting procedure. Communications of the ACM 2(7) : 30 – 32.

[11] Khreisat, L. 2007. Quicksort A Historical Perspective and Empirical Study. International Journal of Computer Science and Network Security 7(12) : 54 – 65.

[12] Williams, J.W.J. 1964. Heapsort. Communications of the ACM 7(6) : 347 – 348.

[13] Cook, C.R. and Kim, D.J. 1980. Best sorting algorithm for nearly sorted lists. Communications of the ACM 23(11) : 620 – 624.

[14] Motzkin, D. 1983. Pracniques : Meansort. Communications of the ACM 26(4) : 250 – 251.

[15] Wainwright, R.L. 1987. Quicksort algorithms with an early exit sorted subfiles. The 15th annual conference on Computer Science 1987 : 183 – 190.

[16] Sarwar, S.M., Sarwar, S.A., Jaragh, M.H.A. and Brandeburg, J. 1996. Engineering quicksort, Elsevier Science 22(1) : 39 – 47.

[17] Hossain, M., Karim, Md.E., Mottalib, Md.A. and Hasan, S. 1998. An Improved Sorting Algorithm. International Conference on Computer and Information Technology 1998 : 220 – 223.

[18] Edmonson, J. 2006. M Pivot Sort – Faster than Quick Sort!. McNair Research Review 4(6) : 59 – 69.

[19] Jidol, J., Sinapiromsaran, K. and Sinthupinyo, S. 2007. A Successive Difference Quicksort. The 4th International Joint Conference on Computer Science and Software Engineering 2007 : 321 – 326.

[20] Weiss, M.A. 1997. Data Structures and Algorithm Analysis in C. CA : Addison-Wesley Press.

[21] Demsar, J. 2006. Statistical Comparisons of Classifiers over Multiple Data Sets. Journal of Machine Learning Research 2006 : 1 – 30.

ศูนย์วิทยทรัพยากร
จุฬาลงกรณ์มหาวิทยาลัย

ศูนย์วิทยทรัพยากร
จุฬาลงกรณ์มหาวิทยาลัย

ภาคผนวก

ภาคผนวก ก
โปรแกรมการเรียงลำดับ

ฟังก์ชันสลับที่

```
void swap(int& x, int& y){
    int temp = x;
    x = y;
    y = temp;
}
```

การเรียงลำดับแบบผสาน (Merge sort)

```
void mergesort(int array[], int temp[], int left, int right){
    if(left < right){
        int mid = (left + right)/2;
        mergesort(array, temp, left, mid);
        mergesort(array, temp, mid+1, right);
        merge(array, temp, left, mid+1, right);
    }
}

void merge(int array[], int temp[], int left, int mid, int right){
    int i, left_end, tmp_pos, size;
    left_end = mid-1;  tmp_pos = left;  size = right - left + 1;

    while(left <= left_end && mid <= right){
        if(array[left] <= array[mid])
            temp[tmp_pos++] = array[left++];
        else
            temp[tmp_pos++] = array[mid++];
    }
    while(left <= left_end)
        temp[tmp_pos++] = array[left++];
    while(mid <= right)
        temp[tmp_pos++] = array[mid++];
    for(i = 0; i < size; i++, right--)
        array[right] = temp[right];
}
```

ฮีปซอร์ต (Heapsort)

```
void heapsort(int array[], int left, int right){
    int i;
    for(i = (right-1)/2; i >= 0; i--)
        siftdown(array, i, right);
    for(i = right; i >= 1; i--){
        swap(array[0], array[i]);
        siftdown(array, 0, i-1);
    }
}

void siftdown(int array[], int root, int bottom){
    int done, maxchild, temp, cur;
    done = 0;  cur = (root*2)+1;
    while( cur <= bottom && !done ){
        if(cur == bottom)
            maxchild = cur;
        else if(array[cur] > array[cur+1])
            maxchild = cur;
        else
            maxchild = cur + 1;
        if(array[root] < array[maxchild]){
            swap(array[root],array[maxchild]);
            root = maxchild;
            cur = (root*2)+1;
        }
        else  done = 1;
    }
}
```

การเรียงลำดับแบบแทรก (Insertion sort)

```
void insertion(int a[],int left,int right){
    int i, j, temp;
    for(i = left+1; i <= right; i++){
        temp = a[i];
        for(j = i; (j > left) && (temp < a[j-1]); j--)
            a[j] = a[j-1];
        a[j] = temp;
    }
}
```

ควิกซอร์ตจากงานวิจัยของ Hoare

```
void HQuicksort(int array[],int left,int right){
    if(right > left){
        int pivot = partition(array,left,right);
        HQuicksort(array,left,pivot-1);
        HQuicksort(array,pivot+1,right);
    }
}

int partition(int array[],int left,int right){
    int i, j, pivot, temp;
    i = left;  j = right-1;
    // Pivot selection is random element
    temp =  (rand()*RAND_MAX)%(right - left + 1) + left;
    pivot = array[temp];
    swap(array[temp],array[right]);
    // Partitioning
    while(true){
        while(array[i] < pivot)  i++;
        while(array[j] >= pivot)  j--;
        if(i >= j)  break;
        swap(array[i],array[j]);
    }
    swap(array[i],array[right]);
    return i;
}
```

ควิกซอร์ตด้วยตัวก่อนหน้าและตัวตามหลังของตัวหลัก (PSQuicksort)

```
void PSQquicksort(int array[],int left,int right){
    // Size of array is small call Insertion sort
    if(right - left <= 9)  insertion(array,left,right);
    else{
        int pivot = PSpartition_Direct/Bubble(array,left,right);
        PSQquicksort(array,left,pivot-2);
        PSQquicksort(array,pivot+2,right);
    }
}
```

การแบ่งกั้นข้อมูลด้วยตัวก่อนหน้าและตัวตามหลังของตัวหลักโดยตรง (PSpartition_Direct)

```
int PSpartition_Direct(int array[],int left,int right){
    int i, j, ind_PD, ind_SC, mid, pivot;
    mid = (left + right)/2;

    // Median of three
    if(array[left] > array[mid])  swap(array[left],array[mid]);
    if(array[left] > array[right])  swap(array[left],array[right]);
    if(array[mid] < array[right])  swap(array[mid],array[right]);
    // Result of method: left is small, mid is big, right is median

    i = left+1;  ind_PD = left;  j = ind_SC = right-1;
    pivot = array[right];

    // Partitioning
    while(true){
        while(array[i] < pivot){
            // Update pivot predecessor
            if(array[i] > array[ind_PD])
                swap(array[i],array[ind_PD]);
            i++;
        }
        while(array[j] >= pivot && i <= j){
            // Update pivot successor
            if(array[j] < array[ind_SC])
                swap(array[j],array[ind_SC]);
            j--;
        }
        if(i >= j)  break;
        swap(array[i],array[j]);
    }
    pivot = i;
    swap(array[i],array[right]);
    j = i - 1;

    swap(array[ind_PD],array[j--]);
    if(i < ind_SC)  swap(array[ind_SC],array[++i]);
    return pivot;
}
```

ศูนย์วิทยทรัพยากร
จุฬาลงกรณ์มหาวิทยาลัย

การแบ่งกั้นข้อมูลด้วยตัวก่อนหน้าและตัวตามหลังของตัวโดยการเรียงลำดับแบบฟอง (PSpartition_Bubble)

```
int PSpartition_Bubble(int array[],int left,int right){
    int i, j, pivot, temp, mid;
    mid = (left + right)/2;

    // Median of three
    if(array[left] > array[mid])  swap(array[left],array[mid]);
    if(array[left] > array[right])  swap(array[left],array[right]);
    if(array[mid] < array[right])  swap(array[mid],array[right]);
    // Result of method: left is small, mid is big, right is median
    swap(array[mid],array[right-1]);

    i = left+1;  j = right-2;  pivot = array[right];

    // Partitioning
    while(true){
        while(array[i] < pivot){
            temp = i-1;
            // Bubble up pivot predecessor
            if(array[i] < array[temp])
                swap(array[i],array[temp]);
            i++;
        }
        while(array[j] >= pivot && i < j){
            temp = j+1;
            // Bubble down pivot successor
            if(array[j] > array[temp])
                swap(array[j],array[temp]);
            j--;
        }
        if(i >= j)  break;
        swap(array[i],array[j]);
    }
    pivot = i;
    temp = i + 1;
    if(array[i] < array[temp])  swap(array[i],array[temp]);
    swap(array[i],array[right]);

    return pivot;
}
```

ควิกซอร์ตจากงานวิจัยของ Sedgewick

```
void SWQuicksort(int array[],int left,int right){
    int cur, L, R, tempL[100], tempR[100], i, j, mid, pivot;

    tempL[0] = left;  tempR[0] = right;  cur = 1;
    do{
        cur--;  L = tempL[cur];  R = tempR[cur];

        // Size of array is small call Insertion sort
        if(R - L <= 9)  insertion(array,L,R);
        else{
            // Median of three
            mid = (L + R)/2;
            if(array[L] > array[R])  swap(array[L],array[R]);
            if(array[L] > array[mid])  swap(array[L],array[mid]);
            if(array[mid] < array[R])  swap(array[mid],array[R]);
            // Result of method: left is small, mid is median, right is big

            i = L;  j = R-1;  pivot = array[R];
            // Partitioning
            while(true){
                while(array[i] < pivot && i <= j)  i++;
                while(array[j] >= pivot && i <= j)  j--;
                if(i >= j) break;
                swap(array[i],array[j]);
            }
            swap(array[i],array[R]);
            j = i-1;  i = i+1;

            if(i < R){
                tempL[cur] = i;
                tempR[cur] = R;
                cur++;
            }
            if(j > L){
                tempL[cur] = L;
                tempR[cur] = j;
                cur++;
            }
        }
    }while(cur > 0);
}
```

ศูนย์วิทยทรัพยากร
จุฬาลงกรณ์มหาวิทยาลัย

ควิกซอร์ตจากงานวิจัยของ Bentley และ Mcilroy

```
void qsort(int array[],int left,int right){
    // Size of array is small call Insertion sort
    if(right - left <= 6) insertion(array,left,right);
    else{
        int s, pl, pm, pr, i, j, k, pivot, ind_pl, ind_pr;
        pl = left;
        pm = (right + left)/2;
        pr = right;
        // Size of array is big, use median of nine
        if(right - left > 40){
            s = (right - left)/8;
            pl = Median3 (array,pl,pl+s,pl+2*s);
            pm = Median3 (array,pm-s,pm,pm+s);
            pr = Median3 (array,pr-2*s,pr-s,pr);
        }
        pm = MedianOfThree (array,pl,pm,pr);
        pivot = array[pm];  swap(array[pm],array[right]);
        i = ind_pl = left;  j = ind_pr = right-1;
        // Partitioning
        while(true){
            while(array[i] <= pivot && j >= i){
                // Find an element is equal to pivot
                if(array[i] == pivot)
                    swap(array[i],array[ind_pl++]);
                i++;
            }
            while(array[j] >= pivot && j >= i){
                // Find an element is equal to pivot
                if(array[j] == pivot)
                    swap(array[j],array[ind_pr--]);
                j--;
            }
            if(i > j)  break;
            swap(array[i],array[j]);
        }

        // Check, the pivot element in left bound
        if(min(ind_pl - left, i - ind_pl) > 0){
            for(k = left; k < ind_pl; k++)
                swap(array[k],array[j--]);
        }
        else{
            if(i - ind_pl == 0)  j = left;
        }
```

```
        // Check, the pivot element in right bound
        if(min(ind_pr - j, right - ind_pr) > 0){
            for(k = right; k > ind_pr; k--)
                swap(array[k],array[i++]);
        }
        else{
            if(right - ind_pr == 0)  i = right;
        }

        if(j - left > 0) qsort(array, left, j);
        if(right - i > 0) qsort(array, i, right);
    }
}

// Function median of three
int Median3(int array[],int left,int mid,int right){
  if(array[left] > array[mid])  swap(array[left],array[mid]);
  if(array[left] > array[right])  swap(array[left],array[right]);
  if(array[mid] > array[right])  swap(array[mid],array[right]);
  return mid;
}
```

ควิกซอร์ตด้วยตัวก่อนหน้าและตัวตามหลังของตัวหลักร่วมกับการแบ่งกั้นข้อมูลแบบสปลิตเอ็น (PSQsort)

```
struct pivots{ int left, right; };

void PSQsort(int array[],int left,int right){
    // Size of array is small call Insertion sort
    if(right - left <= 9)  insertion(array,left,right);
    else{
        struct pivots pivotNew = Split-end_PSpartition(array,left,right);
        if(pivotNew.left - left <= 9)  insertion(array,left,pivotNew.left);
        else  PSQsort(array,left,pivotNew.left);

        if(right - pivotNew.right <= 9)  insertion(array,pivotNew.right,right);
        else  PSsort(array,pivotNew.right,right);
  }
}

struct pivots Split-end_PSpartition(int array[],int left,int right){
    int i, j, k, mid, pivot, PD, SC, i_pivot, i_pd, i_sc;
    struct pivots index;
    mid = (left + right)/2;
```

```
// Median of three
if(array[left] > array[mid])  swap(array[left],array[mid]);
if(array[left] > array[right])  swap(array[left],array[right]);
if(array[mid] < array[right])  swap(array[mid],array[right]);
// Result of method: left is small, mid is big, right is median

// Set values & pointer of pivot, pivot predecessor and pivot successor
pivot = array[right];  PD = array[left];  SC = array[mid];
i = i_pd = left+1;  i_pivot = right-1;  j = i_sc = i_pivot-1;
swap(array[mid],array[i_pivot]);

// Partitioning
while(true){
    while(array[i] < pivot){
        // Find pivot predecessor
        if(array[i] >= PD){
            if(array[i] == PD)
                swap(array[i], array[i_pd++]);
            else{
                PD = array[i];  i_pd = left;
                swap(array[i], array[i_pd++]); }
        }
        i++;
    }

    while(array[j] >= pivot && i <= j){
        // Finding pivot
        if(array[j] == pivot){
            swap(array[j], array[i_pivot--]);
            swap(array[j], array[i_sc--]);
        }
        // Finding pivot successor
        else if(array[j] <= SC){
            if(array[j] == SC)
                swap(array[j], array[i_sc--]);
            else{
                SC = array[j];  i_sc = i_pivot;
                swap(array[j], array[i_sc--]);
            }
        }
        j--;
    }
    if(i >= j)  break;
    swap(array[i], array[j]);
}
j = i - 1;
```

ศูนย์วิทยทรัพยากร
จุฬาลงกรณ์มหาวิทยาลัย

```
    // Move pivot predecessor to correct positions
    for(k = left; k < i_pd; k++)
        swap(array[k],array[j--]);

    // Move pivot to correct positions
    for(k = right; k > i_pivot; k--)
        swap(array[k],array[i++]);

    // Move pivot predecessor to correct positions
    if(array[i] != SC){
        k = i_pivot;
        while(array[k] == SC){
            swap(array[k--],array[i++]);
            if(k <= i)  break;
        }
    }
    else{
        while(array[i] == SC)  i++;
    }

    index.left = j;  index.right = i;
    return index;
}
```

ควิกซอร์ตด้วยตัวประชิดตัวหลัก (APQsort)

```
void APQsort(int array[],int left,int right){
    // Size of array is small call Insertion sort
    if(right - left <= 9)  insertion(array,left,right);
    else{
        int mid, difL, difR;
        struct pivots new_index;

        // Median of three
        mid = (left + right)/2;
        if(array[left] > array[mid])  swap(array[left],array[mid]);
        if(array[left] > array[right])  swap(array[left],array[right]);
        if(array[mid] > array[right])  swap(array[mid],array[right]);
        // Result of method: left is small, mid is median, right is big

        difL = array[mid] - array[left];  difR = array[right] - array[mid];

        if(difR < difL){
            if(difR != 0)  new_index = SC_partition (array,left,mid,right);
            else  new_index = PD_partition (array,left,mid,right);
        }
```

```
        else if(difL < difR){
            if(difL != 0)  new_index = PD_partition (array,left,mid,right);
            else  new_index = SC_partition (array,left,mid,right);
        }
        else{
            if(difL != 0)  new_index = PD_partition (array,left,mid,right);
            else  new_index = Split-end_partition (array,left,mid,right);
        }

        if(new_index.left - left <= 9)  insertion(array,left,new_index.left);
        else  APsort(array,left,new_index.left);

        if(right - new_index.right <= 9)  insertion(array,new_index.right,right);
        else  APsort(array,new_index.right,right);
    }
}
```

การแบ่งกั้นข้อมูลด้วยตัวนำหน้าตัวหลัก (PD partition)

```
struct pivots PD_partition(int array[],int left,int mid,int right){
    int i, j, k, pivot, PD, indL, indR, ipd;
    struct pivots index;

    swap(array[mid],array[right]);
    // Set value and pointer of pivot and pivot predecessor
    i = indL = left+1;  j = indR = right-1;
    pivot = array[right];  PD = array[left];  ipd = left;

    while(true){
        while(array[i] < pivot){
            // Update pivot predecessor
            if(array[i] >= PD){
                if(array[i] == PD)
                    swap(array[i],array[indL++]);
                else{
                    ipd = indL;  PD = array[i];
                    swap(array[i],array[indL++]);
                }
            }  i++;
        }
        while(array[j] >= pivot && i <= j){
            // Finding pivot
            if(array[j] == pivot)
                swap(array[j],array[indR--]);
            j--;
        }
        if(i >= j)  break;
        swap(array[i],array[j]);
    }
```

```
    j = i - 1;

    if(indL <= indR){
        // Move pivot predecessor to the correct position
        if(i != indL){
            for(k = ipd; k < indL; k++)
                swap(array[k],array[j--]);
        }
        else  j = left;

        // Move pivot to the correct position
        k = right;
        while(k > indR){
            while(array[i] == pivot && i <= k)  i++;
            if(i >= k)  break;
            swap(array[k--],array[i++]);
        }
        index.left = j;  index.right = i;
    }
    else{
        // case all elements are equal
        index.left = left;  index.right = right;
    }
    return index;
}
```

การแบ่งกั้นข้อมูลด้วยตัวตามหลังตัวหลัก (SC partition)

```
struct pivots SC_partition(int array[],int left,int mid,int right){
    int i, j, k, pivot, SC, indL, indR, isc;
    struct pivots index;

    swap(array[mid],array[left]);
    //Set value and pointer of pivot and pivot successor
    i = indL = left+1;  j = indR = right-1;
    pivot = array[left];  SC = array[right];  isc = right;

    while(true){
        while(array[i] <= pivot && i <= j){
            // Finding pivot
            if(array[i] == pivot)
                swap(array[i],array[indL++]);
            i++;
        }
        while(array[j] > pivot && i <= j){
            // Update pivot successor
            if(array[j] <= SC){
                if(array[j] == SC)
                    swap(array[j],array[indR--]);
```

```
                else{
                    isc = indR;  SC = array[j];
                    swap(array[j],array[indR--]);
                }
            }
            j--;
        }
        if(i >= j)  break;
        swap(array[i],array[j]);
    }

    if(indL <= indR){
        // Move pivot to the correct position
        k = left;
        while(k < indL){
            while(array[j] == pivot && j >= k)  j--;
            if(j <= k)  break;
            swap(array[k++],array[j--]);
        }

        // Move pivot successor to the correct position
        if(i != isc){
            for(k = isc; k > indR; k--)
                swap(array[k],array[i++]);
        }
        else  i = right;
        index.left = j;  index.right = i;
    }
    else{
        index.left = left;  index.right = right;
    }

    return index;
}
```

การแบ่งกั้นข้อมูลด้วยตัวตามหลังตัวหลัก (Split-end partition)

```
struct pivots Split-end_partition(int array[],int left,int mid,int right){
    int i, j, k, IPL, IPR, pivot;
    struct pivots index;

    // Set value and pointer of pivot
    pivot = array[right];  swap(array[mid],array[right-1]);
    i = IPL = left + 1;  j = IPR = right - 2;

    // Partitioning
    while(true){
        while(array[i] <= pivot && j >= i){
            if(array[i] == pivot)
```

```
                swap(array[i], array[IPL++]);
            i++;
        }
        while(array[j] >= pivot && j >= i){
            if(array[j] == pivot)
                swap(array[j], array[IPR--]);
            j--;
        }
        if(i >= j)  break;
        swap(array[i], array[j]);
    }

    // Move the elements which equal to pivot to correct position
    if(min(IPL - left, i - IPL) > 0){
        for(k = left; k < IPL; k++)
            swap(array[k],array[j--]);
    }
    else{
        if(i - IPL == 0)  j = left;
    }

    if(min(IPR - j, right - IPR) > 0){
        for(k = right; k > IPR; k--){
            swap(array[k],array[i++]);
        }
    }
    else{
        if(right - IPR == 0)  i = right;
    }

    index.left = j;  index.right = i;
    return index;
}
```

ศูนย์วิทยทรัพยากร
จุฬาลงกรณ์มหาวิทยาลัย

ภาคผนวก ข
ผลการทดลอง

ผลการทดสอบประสิทธิภาพของ PSQuicksort

Sizes	Mergesort	Heapsort	HQuicksort	PSQuicksort_D	PSQuicksort_B	SWQuicksort
10000	0.0036830	0.0047315	0.0029142	0.0024836	0.0042744	0.0021235
30000	0.0119137	0.0160185	0.0094400	0.0082703	0.0147984	0.0070840
100000	0.0428059	0.0607545	0.0351396	0.0298896	0.0563094	0.0259684

เวลาการประมวลผลเฉลี่ย

Sizes	Mergesort	Heapsort	HQuicksort	PSQuicksort_D	PSQuicksort_B	SWQuicksort
10000	407,685	485,739	266,270	317,403	309,868	177,677
30000	1,363,060	1,646,380	897,729	1,094,460	1,075,390	599,787
100000	5,074,190	6,189,120	3,466,000	4,185,100	4,130,840	2,240,760

จำนวนครั้งที่ใช้เปรียบเทียบเฉลี่ย

Sizes	Mergesort	Heapsort	HQuicksort	PSQuicksort_D	PSQuicksort_B	SWQuicksort
10000	133,616	124,201	38,339	39,176	136,781	38,252
30000	447,232	419,876	126,190	128,705	477,295	125,883
100000	1,668,930	1,574,900	468,961	436,551	1,811,020	426,412

จำนวนครั้งที่ใช้สลับที่เฉลี่ย

Sizes	Mergesort	Heapsort	HQuicksort	PSQuicksort_D	PSQuicksort_B	SWQuicksort
10000	19,999	0	13,568	2,406	2,406	0
30000	59,999	0	40,870	7,220	7,228	0
100000	199,999	0	148,801	24,338	24,529	0

จำนวนครั้งที่เรียกฟังก์ชันเวียนเกิดเฉลี่ย

ศูนย์วิทยทรัพยากร จุฬาลงกรณ์มหาวิทยาลัย

ผลการทดสอบประสิทธิภาพของ PSQsort

Sizes	Method	Compare	Swap	Recursive	Time(s)
10000	SWQuicksort	177,677	38,252	0	0.0021235
	qsort	332,400	34,049	8,342	0.0027316
	PSQsort	313,561	41,500	1,020	0.0026024
30000	SWQuicksort	599,787	125,883	0	0.0070840
	qsort	1,131,770	113,927	24,707	0.0089912
	PSQsort	1,083,580	135,898	3,046	0.0085569
100000	SWQuicksort	2,240,760	426,412	0	0.0259684
	qsort	4,046,070	435,956	42,426	0.0313685
	PSQsort	3,929,010	477,164	7,639	0.0300950

การประมวลผลเฉลี่ยสำหรับข้อมูลแบบที่หนึ่ง

Unsorted Ratio	Method	Compare	Swap	Recursive	Time(s)
0	SWQuicksort	422,773	8,190	0	0.0023400
	qsort	796,086	8,190	8,191	0.0034400
	PSQsort	95,285,300	113,166	9,136	0.2706600
0.01	SWQuicksort	3,055,370	58,956	0	0.0138055
	qsort	966,789	43,767	33,280	0.0056489
	PSQsort	8,936,160	103,886	4,831	0.0318353
0.03	SWQuicksort	1,615,520	64,819	0	0.0081909
	qsort	978,725	50,210	32,673	0.0059560
	PSQsort	4,130,310	99,807	4,340	0.0167076
0.1	SWQuicksort	954,877	74,637	0	0.0058722
	qsort	992,949	57,974	31,608	0.0063992
	PSQsort	2,154,130	95,773	3,851	0.0103522
0.3	SWQuicksort	744,410	89,333	0	0.0055727
	qsort	1,025,060	72,107	29,680	0.0071567
	PSQsort	1,338,500	102,102	3,382	0.0080226

การประมวลผลเฉลี่ยสำหรับข้อมูลแบบที่สอง

ศูนย์วิทยทรัพยากร
จุฬาลงกรณ์มหาวิทยาลัย

Unsorted Ratio	Method	Compare	Swap	Recursive	Time(s)
0	SWQuicksort	467,770	23,190	0	0.0029700
	qsort	1,173,180	60,394	20,377	0.0062300
	PSQsort	80,343,300	131,506	8,304	0.2982700
0.01	SWQuicksort	2,785,270	70,403	0	0.0128148
	qsort	1,009,750	56,329	33,011	0.0057673
	PSQsort	7,952,980	117,288	5,060	0.0297323
0.03	SWQuicksort	1,579,790	77,014	0	0.0081704
	qsort	1,018,690	62,752	32,757	0.0060806
	PSQsort	4,008,150	113,000	4,384	0.0166721
0.1	SWQuicksort	948,546	88,210	0	0.0060681
	qsort	1,042,550	73,941	31,615	0.0066120
	PSQsort	2,043,650	109,634	3,850	0.0101015
0.3	SWQuicksort	767,823	105,615	0	0.0060734
	qsort	1,088,980	92,087	29,803	0.0075689
	PSQsort	1,298,910	117,324	3,385	0.0080481

การประมวลผลเฉลี่ยสำหรับข้อมูลแบบที่สาม

Unrepeated	Method	Compare	Swap	Recursive	Time(s)
0.001	SWQuicksort	15,571,100	65,033	0	0.0648253
	qsort	349,170	68,798	30	0.0029598
	PSQsort	316,701	90,765	15	0.0032669
0.01	SWQuicksort	2,221,710	86,156	0	0.0118118
	qsort	626,895	92,811	300	0.0047133
	PSQsort	586,044	111,360	146	0.0049779
0.03	SWQuicksort	1,225,800	91,702	0	0.0081525
	qsort	768,823	105,892	900	0.0057887
	PSQsort	727,358	121,872	437	0.0059075
0.1	SWQuicksort	788,523	91,901	0	0.0067437
	qsort	934,267	119,524	3,003	0.0071844
	PSQsort	885,618	128,133	1,212	0.0070584
0.3	SWQuicksort	762,706	113,884	0	0.0070607
	qsort	1,049,220	118,683	11,440	0.0081610
	PSQsort	1,007,160	130,418	2,198	0.0079900

การประมวลผลเฉลี่ยสำหรับข้อมูลแบบที่สี่

ศูนย์วิทยทรัพยากร
จุฬาลงกรณ์มหาวิทยาลัย

ประวัติผู้เขียนวิทยานิพนธ์

ชื่อ – สกุล	นายศิวัช เรืองพิภพ
วัน เดือน ปีเกิด	10 ตุลาคม 2528
ประวัติการศึกษา	2549 จบการศึกษาปริญญาวิทยาศาสตร์บัณฑิต สาขาคณิตศาสตร์ประยุกต์ คณะวิทยาศาสตร์ประยุกต์ สถาบันเทคโนโลยีพระจอมเกล้าพระนครเหนือ

ศูนย์วิทยทรัพยากร
จุฬาลงกรณ์มหาวิทยาลัย