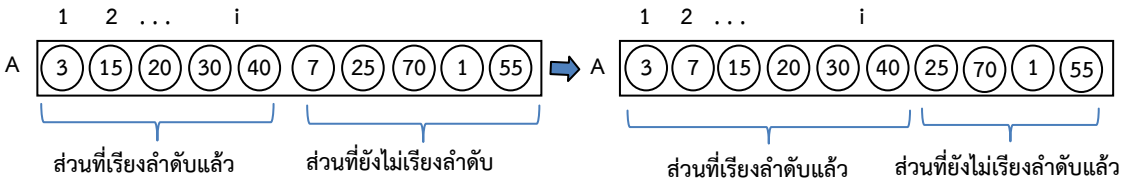
บทที่ 9

การเรียงลำดับ

ปัญหาการเรียงลำดับข้อมูลเป็นปัญหาพื้นฐานสำหรับผู้พัฒนาโปรแกรมที่ต้องจัดการ เนื่องจากในการแก้ปัญหาที่สลับซับซ้อนต่างๆ ล้วนต้องอาศัยการจัดเรียงข้อมูลทั้งหมด ไม่ว่าจะเป็นปัญหาการจัดตารางงาน (Job Scheduling) ปัญหาการค้นหาเส้นทางที่สั้นที่สุด เป็นต้น ซึ่งอัลกอริทึมที่ใช้ในการเรียงลำดับข้อมูลมีจำนวนมาก แต่ละวิธีจะมีหลักการและเวลาในการจัดเรียงต่างกัน การเรียงลำดับในแต่ละวิธีจะใช้เพื่อการจัดเรียงข้อมูลจากน้อยไปมากหรือมากไปน้อยก็ได้

6.1 การเรียงลำดับแบบแทรก

การเรียงลำดับแบบแทรก (Insertion Sort) จะใช้หลักการแยกข้อมูลเป็นสองส่วน คือ ส่วนที่เรียงลำดับแล้วกับส่วนที่ยังไม่ได้เรียงลำดับ ดังรูปที่ 9.1 ข้อมูลในอาร์เรย์ A[0..i] คือ ส่วนของข้อมูลที่เรียงลำดับแล้ว และข้อมูลในอาร์เรย์ A[i+1 .. N-1] คือ ส่วนของข้อมูลที่ยังไม่เรียงลำดับ วิธีการจัดเรียงข้อมูลจะใช้หลักการแทรก โดยนำค่าข้อมูลในส่วนของข้อมูลที่ยังไม่เรียงลำดับไปวางแทรกในอาร์เรย์ในส่วนของข้อมูลที่เรียงลำดับแล้วทีละค่า รูปที่ 9.1 (ก) แสดงตัวอย่างกรณี i = 4 และ N=10 ข้อมูล A[0..4] เป็นส่วนที่เรียงลำดับแล้ว และข้อมูลที่ A[5..9] เป็นส่วนข้อมูลที่ยังไม่เรียงลำดับ เมื่อนำข้อมูล A[5]=7 ไปเรียงแทรกไว้ในส่วนที่เรียงลำดับแล้ว พบว่า 7 จะถูกแทรกไว้ที่ตำแหน่ง A[1] ดังผลลัพธ์ในรูปที่ 9.1 (ข) โดยข้อมูลในส่วนที่ยังไม่เรียงลำดับจะถูกนำไปเรียงลำดับทีละค่าด้วยหลักการแทรก จนไม่เหลือข้อมูลในส่วนที่ยังไม่ได้เรียงลำดับ จะได้ข้อมูลที่เรียงลำดับจากน้อยไปมากครบทุกค่า



(ก) แบ่งข้อมูลเป็น 2 ส่วน (ข) หลังการเรียงลำดับข้อมูล 5

รูปที่ 9.1 หลักการเรียงลำดับแบบแทรก

จากหลักการของการเรียงลำดับแบบแทรก สามารถแสดงตัวอย่างการเรียงลำดับจากน้อยไปมากด้วยหลักการแบบแทรกในกรณีมีข้อมูลทั้งหมด 5 ค่า ดังตารางที่ 9.1 ซึ่งมีรายละเอียดดังนี้

- เริ่มต้นจะกำหนดให้ค่าตัวแรกในอาร์เรย์ A[i] คือ 8 เป็นส่วนข้อมูลที่เรียงลำดับแล้ว ส่วนข้อมูลในอาร์เรย์ A[i+1..N-1] คือ 3 9 5 1 เป็นส่วนของข้อมูลที่ยังไม่ได้เรียงลำดับ
- ในแถวที่ 1 ส่วนข้อมูลที่เรียงลำดับแล้ว คือ A[0] ดังนั้น i=0 จะเรียงลำดับโดยนำข้อมูลที่ A[i+1] คือ 3 มาเปรียบเทียบกับข้อมูลในส่วนที่เรียงลำดับ (ขณะนี้ คือ 8) ทีละค่า พบว่า 3 น้อยกว่า 8 จึงต้องนำ 8 ถอยไปไว้ในตำแหน่ง A[1] และนำ 3 ใส่แทรกในตำแหน่ง A[0]
- ในแถวที่ 2 ส่วนข้อมูลที่เรียงลำดับแล้ว คือ A[0..1] ดังนั้น i=1 จะเรียงลำดับโดยนำข้อมูลที่ A[i+1] คือ 9 มาเปรียบเทียบกับข้อมูลในส่วนที่เรียงลำดับ (ขณะนี้ คือ 3 8) ทีละค่า พบว่า

9 มากกว่า 8 ดังนั้น 9 จึงวางอยู่ตำแหน่งเดิม ซึ่งเป็นกรณีที่ไม่มีการเปลี่ยนแปลงข้อมูลในส่วนที่เรียงลำดับแล้วเลย

- ในแถวที่ 3 ส่วนข้อมูลที่เรียงลำดับแล้ว คือ A[0..2] ดังนั้น i=2 จะเรียงลำดับโดยนำข้อมูลที่ A[i+1] คือ 5 มาเปรียบเทียบกับข้อมูลในส่วนที่เรียงลำดับ (ขณะนี้ คือ 3 8 9) ทีละค่า พบว่า 5 น้อยกว่า 9 จึงต้องนำ 9 ถอยไปไว้ในตำแหน่ง A[3] และ 5 น้อยกว่า 8 จึงต้องนำค่า 8 ถอยไปไว้ในตำแหน่ง A[2] แต่ 5 มากกว่า 3 ดังนั้นจึงหยุดการเปรียบเทียบแล้วนำ 5 ใส่แทรกในตำแหน่ง A[1]
- ในแถวที่ 4 ส่วนข้อมูลที่เรียงลำดับแล้ว คือ A[0..3] ดังนั้น i=3 จะเรียงลำดับโดยนำข้อมูลที่ A[i+1] คือ 1 มาเปรียบเทียบกับข้อมูลในส่วนที่เรียงลำดับ (ขณะนี้ คือ 3 5 8 9) ทีละค่า พบว่า 1 น้อยกว่า 9 8 5 และ 3 จึงต้องนำ 9 8 5 และ 3 ถอยไปไว้ในตำแหน่ง A[4] A[3] A[2] และ A[1] ตามลำดับการเปรียบเทียบ กรณีนี้เป็นการเปรียบเทียบกับข้อมูลทุกตัวในส่วนที่เรียงลำดับแล้ว จากนั้นจึงนำ 1 ใส่แทรกในตำแหน่ง A[0]

ตารางที่ 9.1 ตัวอย่างการเรียงลำดับแบบแทรก

ลำดับ	ข้อมูลในอาร์เรย์ A	i+1	A[i+1]	ลำดับการแทรก
1	8 3 9 5 1	1	3	8 3 9 5 1
2	3 8 9 5 1	2	9	3 8 9 5 1
3	3 8 9 5 1	3	5	3 8 5 9 1 3 5 8 9 1
4	3 5 8 9 1	4	1	3 5 8 1 9 3 5 1 8 9 3 1 5 8 9 1 3 5 8 9

ตัวอย่างคำสั่งภาษาซีของการเรียงลำดับจากน้อยไปมากด้วยหลักการแบบแทรก

```
voidinsertion_sort(intA[], intN) {
    inti, j, index;
    for (i=1; i<N; i++) {
(1)     key = A[i];
        j = i;
(2)     while ((j > 0) && (A[j-1] > index))  {
            A[j] = A[j-1];
            j = j – 1;
        }
(3)     A[j] = key;
    }
}
```

จัดเรียงทีละค่า (ตั้งแต่ i=1 ถึง N-1) ดังนี้
(1) กำหนดค่า key = A[i]
(2) เปรียบเทียบค่า key กับ A[j] (j=i-1..0) ทีละค่าข้อมูลในอาร์เรย์ A[0...i-1] เพื่อค้นหาตำแหน่ง j ที่เหมาะสมในการแทรกข้อมูล
(3) กำหนดค่า A[j] = key
วนทำซ้ำทั้ง 3 ข้อจนครบทุกค่า i

คำสั่งข้างต้นเป็นการเรียงลำดับจากน้อยไปมาก หากต้องการเรียงลำดับจากมากไปน้อยให้เปลี่ยนวิธีการเปรียบเทียบเงื่อนไขในการค้นหาตำแหน่ง j ที่เหมาะสมในการแทรกข้อมูลในคำสั่งในข้อ 2

6.2 การเรียงลำดับแบบเลือก

การเรียงลำดับแบบเลือก (Selection Sort) จะใช้หลักการเลือกข้อมูลที่น้อยที่สุดหรือมากที่สุดไปวางไว้ในตำแหน่งที่เหมาะสมในอาร์เรย์ทีละค่า และวนทำซ้ำจนกว่าข้อมูลทุกตัวจะวางอยู่ในตำแหน่งที่เหมาะสม กรณีต้องการเรียงลำดับข้อมูลจากน้อยไปมาก อาจใช้วิธีการเลือกค่ามากที่สุดจากอาร์เรย์ในตำแหน่งที่ A[0], A[1], ... , A[N-1] ด้วยการเปรียบเทียบข้อมูลทีละค่าเพื่อหาว่าตำแหน่งใดเก็บค่ามากที่สุด (สมมุติว่าเป็นตำแหน่งที่ max) จากนั้นจะสลับค่าข้อมูล A[max] กับ A[N-1] ดังนั้น A[N-1] เก็บค่ามากที่สุดและถือว่าอยู่ในตำแหน่งที่เหมาะสมคือเรียงลำดับแล้ว จากนั้นให้วนทำซ้ำโดยเลือกค่ามากที่สุดตัวถัดไปจากข้อมูลในอาร์เรย์ A[0], A[1], ... , A[N-2] แล้วนำค่ามากที่สุดนั้นไปสลับกับค่า A[N-2] จะพบว่าข้อมูล A[N-2] เก็บค่ามากที่สุดตัวถัดไป ในรอบที่สองนี้ A[N-2] และ A[N-1] เก็บข้อมูลในตำแหน่งที่เหมาะสม คือ เรียงลำดับแล้ว เมื่อวนทำซ้ำไปเรื่อยๆ จนครบทุกค่าในอาร์เรย์จะได้ข้อมูลที่เรียงลำดับจากน้อยไปมากแล้วทั้งหมด

ตารางที่ 9.2 แสดงตัวอย่างขั้นตอนการเรียงลำดับจากน้อยไปมากด้วยหลักการเรียงลำดับแบบเลือกด้วยของข้อมูล 5 ค่า มีรายละเอียดดังนี้

- ในแถวที่ 1 หาตำแหน่งที่มากที่สุดในอาร์เรย์ตำแหน่งที่ A[0..4] พบว่าคือตำแหน่ง 2 (ค่า 9) จึงสลับ A[2] กับ A[4] แสดงว่า 9 อยู่ในตำแหน่งสุดท้ายของอาร์เรย์ และเรียงลำดับเสร็จแล้ว 1 ค่า
- ในแถวที่ 2 หาตำแหน่งที่มากที่สุดในอาร์เรย์ตำแหน่งที่ A[0..3] พบว่าคือตำแหน่ง 0 (ค่า 8) จึงสลับ A[0] กับ A[3] ดังนั้นขณะนี้ 8 9 เก็บในอาร์เรย์แบบเรียงลำดับแล้ว
- ในแถวที่ 3 หาตำแหน่งที่มากที่สุดในอาร์เรย์ตำแหน่งที่ A[0..2] พบว่าคือตำแหน่ง 0 (ค่า 5) จึงสลับ A[0] กับ A[2] ดังนั้นขณะนี้ 5 8 9 เก็บในอาร์เรย์แบบเรียงลำดับแล้ว
- ในแถวที่ 4 หาตำแหน่งที่มากที่สุดในอาร์เรย์ตำแหน่งที่ A[0..1] พบว่าคือตำแหน่ง 1 (ค่า 3) ซึ่งเป็นกรณีที่ไม่เกิดการสลับค่าในอาร์เรย์ ทำให้ขณะนี้ 3 5 8 9 เก็บในอาร์เรย์แบบเรียงลำดับแล้ว เนื่องจากเหลือค่า 1 ในช่อง A[0] เพียงตัวเดียว จึงไม่ต้องเลือกค่าสูงสุดแล้ว ดังนั้นข้อมูลทุกตัวในอาร์เรย์เรียงลำดับครบแล้ว

ตารางที่ 9.2 ตัวอย่างการเรียงลำดับแบบเลือกด้วยการหาค่ามากที่สุด

ลำดับ	ข้อมูลในอาร์เรย์ A	i	ตำแหน่งmax เก็บค่ามากสุดใน A[0..i]	หลังการสลับค่า A[max] กับ A[i]
1	8 3 9 5 1	4	2	8 3 1 5 9
2	8 3 1 5 9	3	0	5 3 1 8 9
3	5 3 1 8 9	2	0	1 3 5 8 9
4	1 3 5 8 9	1	1	1 3 5 8 9

การเรียงลำดับข้อมูลจากน้อยไปมาก อาจใช้วิธีการเลือกค่าน้อยที่สุดจากอาร์เรย์ในตำแหน่งที่ A[0], A[1], ... , A[N-1] ด้วยการเปรียบเทียบข้อมูลทีละค่าเพื่อหาว่าตำแหน่งใดเก็บค่าน้อยที่สุด (สมมุติว่าเป็นตำแหน่งที่ min) จากนั้นจะสลับค่าข้อมูลน้อยสุด A[min] กับ A[0] ดังนั้น A[0] เก็บค่าน้อยที่สุดและถือว่าอยู่ในตำแหน่งที่เหมาะสม คือ เรียงลำดับแล้ว จากนั้นให้วนทำซ้ำโดยเลือกค่าน้อยที่สุดตัวถัดไปวางไว้ในตำแหน่งที่ A[1], A[2], ... , A[N-1] ทีละค่า จะได้ผลลัพธ์จากการเรียงลำดับ

แบบเลือกเป็นข้อมูลที่เรียงลำดับจากน้อยไปมากเช่นเดียวกัน ดังตัวอย่างในตารางที่ 9.3 แสดงการเรียงลำดับแบบเลือกด้วยการหาค่าน้อยที่สุด

ตารางที่ 9.3 ตัวอย่างการเรียงลำดับแบบเลือกด้วยการหาค่าน้อยที่สุด

ลำดับ	ข้อมูลในอาร์เรย์ A	i	ตำแหน่ง min เก็บค่าน้อยสุดใน A[i..n]	หลังการสลับค่า A[min] กับ A[i]
1	8 3 9 5 1	0	4	1 3 9 5 8
2	1 3 9 5 8	1	1	1 3 9 5 8
3	1 3 9 5 8	2	0	1 3 5 9 8
4	1 3 5 9 8	3	1	1 3 5 8 9

ตัวอย่างภาษาซี ของการเรียงลำดับแบบเลือกด้วยวิธีการเลือกค่าน้อยที่สุดไปวางไว้ในตำแหน่งที่เหมาะสมทีละค่าจนครบทุกตัวข้างต้น มีดังนี้

```
voidselection_sort(intA[], int N){
    for (i = 0; i<N-1; i++)  {
(1)   min = i;
(2)   for (j = i+1; j <N; j++)
          if (A[j] <A[min])
              min = j;
      temp = A[i];
      A[i] = A[min];
(3)   A[min] = temp;
    }
}
```

เลือกค่าน้อยที่สุดไปวางไว้ในตำแหน่งที่เหมาะสมทีละค่า (ตั้งแต่ i=0 ถึง N-1) ดังนี้

(1) กำหนดให้ min = i

(2) เปรียบเทียบค่า A[min] กับ A[j] (j=i+1..N-1) ทีละค่าข้อมูลในอาร์เรย์ A[0...i-1] เพื่อเพื่อหาตำแหน่ง min สำหรับเก็บตำแหน่งที่จะวางค่าน้อยที่สุด

(3) สลับค่า A[min] กับ A[i]

6.3 การเรียงลำดับแบบฟอง

การเรียงลำดับแบบฟอง (Buble Sort) จะใช้หลักการเปรียบเทียบ 2 ค่าที่อยู่ติดกัน (A[j] กับ A[j+1]) ว่าอยู่ในลำดับที่เหมาะสมหรือไม่ หากอยู่ในตำแหน่งที่ไม่เหมาะสมจะทำการสลับค่า เช่น กรณีเรียงจากน้อยไปมาก หาก A[j] มากกว่า A[j+1] ก็จะสลับตำแหน่งกันเพื่อให้ค่ามากไปอยู่ด้านหลัง เป็นต้น โดยจะสลับค่าทีละคู่ตั้งแต่ A[0] กับ A[1] A[1] กับ A[2] จนถึง A[N-2] กับ A[N-1] เมื่อทำครบทุกคู่จะพบว่าค่ามากสุดจะถูกสลับไปวางอยู่ในตำแหน่งสุดท้ายในอาร์เรย์ ถือเป็นการเรียงข้อมูลเสร็จ 1 ค่า

ตารางที่ 9.4 แสดงตัวอย่างการเรียงลำดับ ข้อมูลจากน้อยไปมากด้วยหลักการเรียงลำดับ แบบฟองของข้อมูลจำนวน 5 ค่า พบว่าในแถวที่ 1 เพื่อให้ค่ามากที่สุดไปวางในตำแหน่งที่ n (ตำแหน่ง A[4]) จะเริ่มพิจารณาการสลับค่าทีละคู่ระหว่าง A[j] กับ A[j+1] เมื่อ i=0..3 จะพบว่า 9 อยู่ในตำแหน่งสุดท้ายของอาร์เรย์ และเรียงลำดับเสร็จแล้ว 1 ค่าจากนั้นจะพิจารณาการสลับค่าทีละคู่ใหม่ก็จะได้ข้อมูลที่มีค่ามากที่สุดตัวถัดไป คือ 8 วางไว้ในตำแหน่ง A[3] จะพบว่า 8 และ 9 เรียงลำดับแล้ว ดังตัวอย่างในแถวที่ 2 ทำเช่นนี้ไปเรื่อยๆ จะได้ข้อมูลทุกค่าจะถูกเรียงลำดับจากน้อยไปมาก ดังผลลัพธ์ในแถวสุดท้ายของตารางที่ 9.4

ตารางที่ 9.4 ตัวอย่างการเรียงลำดับแบบฟอง

ลำดับ	ข้อมูลในอาร์เรย์ A	i	j	พิจารณาค่า A[j] กับ A[j+1], j=0..i-1	ผลลัพธ์การสลับค่า
1	8 3 9 51	4	0	สลับค่า 8 และ 3	8 3 9 5 1
			1	ไม่สลับค่า	3 8 9 5 1
			2	สลับค่า 9 และ 5	3 8 5 9 1
			3	สลับค่า 9 และ 1	3 8 5 1 9
2	3 8 5 1 9	3	0	ไม่สลับค่า	3 8 5 1 9
			1	สลับค่า 8 และ 5	3 5 8 1 9
			2	สลับค่า 8 และ 1	3 5 1 8 9
3	3 5 1 8 9	2	0	ไม่สลับค่า	3 5 1 8 9
			1	สลับค่า 5 และ 1	3 1 5 8 9
4	3 1 5 8 9	1	0	สลับค่า 3และ 1	1 3 5 8 9

ตัวอย่างภาษาซี ของการเรียงลำดับแบบฟองด้วยวิธีการสลับเพื่อให้ค่ามากที่สุดไปวางไว้ในตำแหน่งที่เหมาะสมทีละค่า มีดังนี้

```
voidbubble_sort(intA[], intN) {
    inti, j, temp;
(1) for (i = N-1; i> 0; i--)
    (2) for (j = 0; j <i ; j++)
            if (A[j] >A[ j+1 ]) {
                temp = A[ j ];
                A[ j ] = A[ j+1 ];
                A[ j+1 ] = temp;
            }
}
```

(3) เริ่มต้น i=N–1 (i=N–1 ถึง 1)

(4) ทำการสลับค่ามากไปวางไว้ด้านหลังทีละค่าในตำแหน่งที่เหมะสม โดยเปรียบเทียบคู่ข้อมูลระหว่าง A[j] กับ A[j+1] (j=0 ... i–1) จนครบทุกคู่ จะได้ค่ามากสุดจะอยู่ใน A[i]

วนทำซ้ำข้อ (2) จนครบทุกค่า i

สำหรับวิธีการเรียงลำดับข้อมูลแบบฟองเพื่อเรียงข้อมูลจากมากไปน้อย อาจใช้วิธีการสลับค่าน้อยสุดไปวางไว้ในตำแหน่งที่เหมาะสมทีละค่าจนกว่าจะครบทุกค่าข้อมูลก็ได้

6.4 การเรียงลำดับแบบผสาน

การเรียงลำดับแบบผสาน (Merge Sort) เป็นวิธีการเรียงลำดับที่นิยมใช้วิธีหนึ่ง และเหมาะกับข้อมูลขนาดใหญ่ โดยอาศัยเทคนิคในการพัฒนาอัลกอริทึมที่เรียกว่าวิธีการแบ่งแยกและเอาชนะ (Divide and Conquer) ดังนั้นขั้นตอนของอัลกอริทึมในการเรียงลำดับแบบผสานจึงแบ่งเป็น 2 ขั้นตอนหลัก คือ ขั้นตอนการแบ่งแยกข้อมูล และขั้นตอนการผสานเพื่อเรียงลำดับข้อมูล

รูปที่ 9.2 แสดงตัวอย่างการเรียงลำดับแบบผสาน ในขั้นตอนแรกจะต้องแบ่งข้อมูลออกเป็นสองส่วนก่อน คือ ฝั่งซ้ายและขวา ซึ่งจะแบ่งไปเรื่อย ๆ จนเหลือเพียงค่าเดียวในแต่ละฝั่ง จากนั้น จะนำข้อมูลทีละ 2 ค่า มาเรียงลำดับด้วยการผสาน ข้อมูลทั้งสองฝั่ง เรียกขั้นตอนการผสาน (Merge)

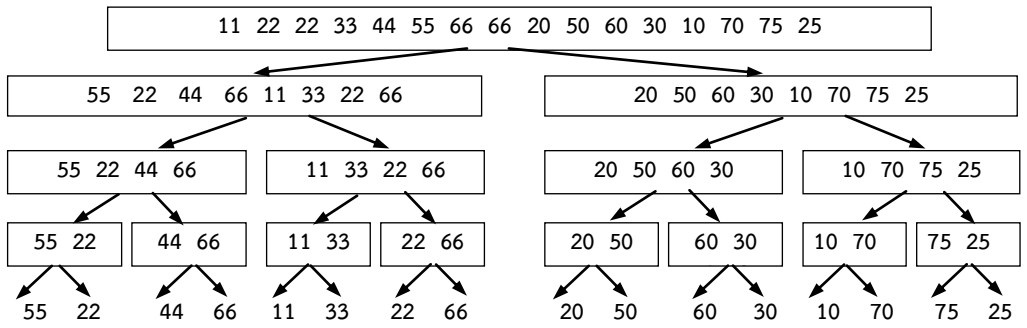
จากนั้นทำการผสาน อีกครั้งหนึ่งด้วยการนำ 2 ค่าที่เรียงแล้วไป ผสานกับอีก 2 ค่าของอีกฝั่งหนึ่ง ก็จะได้ข้อมูลรวม 4 ค่าที่เรียงลำดับแล้ว จากนั้นนำ 4 ค่าที่เรียงลำดับไปกับรวมอีก 4 ค่าที่เรียงลำดับแล้วของอีกฝั่งหนึ่งก็จะได้ 8 ค่าที่เรียงลำดับ ทำเช่นนี้ไปเรื่อยๆ จนครบ n ค่า

ตัวอย่างคำสั่งภาษาซีสำหรับการเรียงลำดับแบบผสาน มีดังนี้

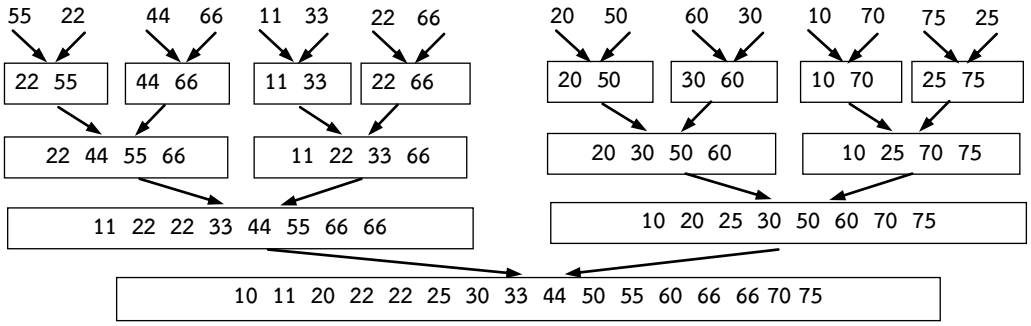
```
void merge_sort(int numbers[], int temp[], int left, int right) {
    int mid;
    if (right > left) {
      mid = (right + left) / 2;
(1)   merge_sort(numbers, temp, left, mid);
      merge_sort(numbers, temp, mid+1, right);
(2)   merge(numbers, temp, left, mid+1, right);
    }
}
```

(1) แบ่งแยกข้อมูลออกเป็น 2 ส่วน คือ ฝั่งซ้ายและฝั่งขวา ซึ่งจะทำการแบ่งซ้ำไปเรื่อยๆ ด้วยหลักการเรียกซ้ำ (Recursive) จนเหลือข้อมูลเพียงฝั่งละ 1 ค่า

(2) ทำการผสานข้อมูลทั้งสองฝั่งเพื่อเรียงลำดับข้อมูล



ก) ขั้นตอนการแบ่งข้อมูลเป็น 2 ส่วน



ข) ขั้นตอนการผสาน (Merge)

รูปที่ 9.2 หลักการเรียงลำดับแบบผสาน

สำหรับขั้นตอนการผสาน จะมีการเรียกใช้ฟังก์ชัน merge() ดังตัวอย่างในรูปที่ 9.3 โดยสามารถเขียนเป็นคำสั่งภาษาซีได้ดังนี้

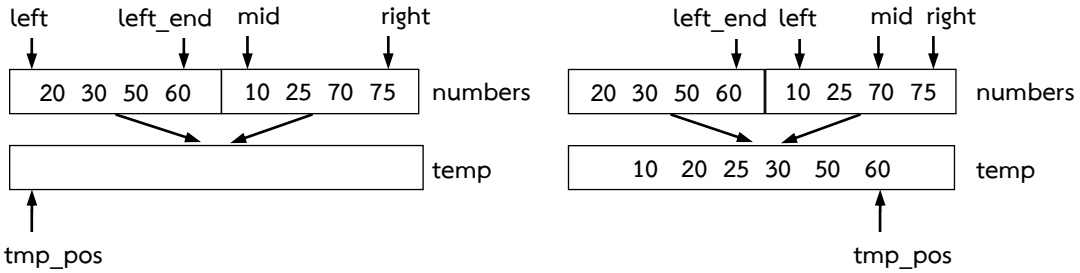
```
void merge(int numbers[], int temp[], int left, int mid, int right) {
    inti, left_end, num_elements, tmp_pos;
    left_end = mid - 1;
    tmp_pos = left;
    num_elements = right - left + 1;
    while ((left <= left_end) && (mid <= right))  {
        if (numbers[left] <= numbers[mid])  {
(1)         temp[tmp_pos] = numbers[left];
            tmp_pos = tmp_pos + 1;
            left = left +1;
        }
        else {
            temp[tmp_pos] = numbers[mid];
            tmp_pos = tmp_pos + 1;
            mid = mid + 1;
            }
    }
    while (left <= left_end) {
        temp[tmp_pos] = numbers[left];
(2)     left = left + 1;
        tmp_pos = tmp_pos + 1;
    }
    while (mid <= right) {
        temp[tmp_pos] = numbers[mid];
(3)     mid = mid + 1;
        tmp_pos = tmp_pos + 1;
    }
    for (i=0; i<num_elements; i++) {
(4)     numbers[right] = temp[right];
        right = right - 1;
    }
}
```

(1) เริ่มต้นการผสานด้วยการกำหนดค่าตัวชี้สำหรับเป็น index ของอาร์เรย์ส่วนที่ 1 (ฝั่งซ้าย) และส่วนที่ 2 (ฝั่งขวา) เพื่อใช้ในการเข้าถึงข้อมูลที่จะนำมาเปรียบเทียบกันจากทั้งสองอาร์เรย์ ดังรูปที่ 9.3 (ก)

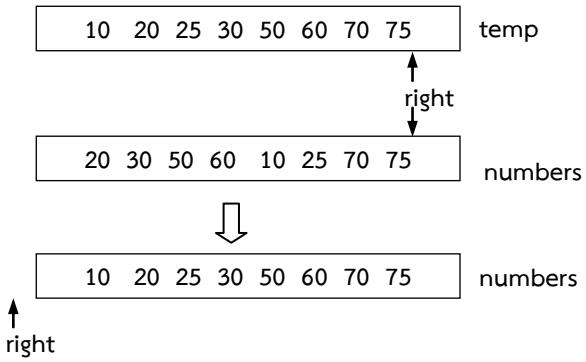
(2) หลังการเปรียบเทียบข้อมูลจากทั้งสองส่วนหากเหลือข้อมูลในส่วนที่ 1 (ฝั่งซ้าย) ให้นำข้อมูลที่เหลือไปวางต่อท้ายไว้ในอาร์เรย์ temp

(3) หลังการเปรียบเทียบข้อมูลจากทั้งสองส่วน หากเหลือข้อมูลในส่วนที่ 2 (ฝั่งขวา) ดังตัวอย่างในรูปที่ 9.3 (ข) ให้นำข้อมูลที่เหลือไปวางต่อท้ายไว้ในอาร์เรย์ temp

(4) เมื่อนำข้อมูลจากทั้งสองส่วนมาจัดเรียงในอาร์เรย์ temp เรียบร้อยแล้ว จะสำเนากลับไปวางไว้ในอาร์เรย์ numbers ในตำแหน่งที่ left .. right แบบเรียงจากน้อยไปมาก ดังรูปที่ 9.3 (ค)



(ก) การกำหนดตัวแปรเริ่มต้นก่อนการผสาน **(ข) หลังการเปรียบเทียบเพื่อรวมข้อมูลทั้ง 2 ส่วน**



(ค) ย้ายข้อมูลจากอาร์เรย์ temp ไปวางอาร์เรย์ numbers

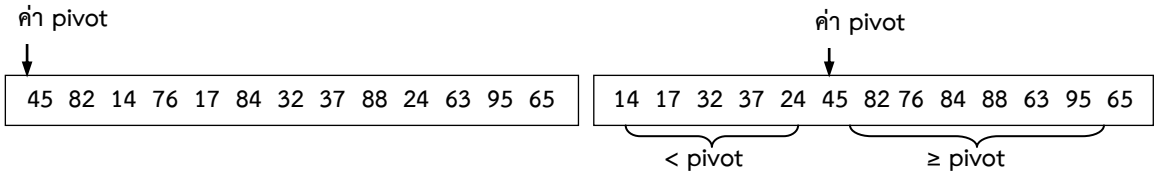
รูปที่ 9.3 ขั้นตอนการผสานระหว่างข้อมูล 2 ส่วนแบบเรียงลำดับ

6.5 การเรียงลำดับแบบเร็ว

การเรียงลำดับแบบเร็ว (Quick Sort) เป็นการเรียงลำดับข้อมูลโดยการแบ่งข้อมูลเป็นส่วนย่อยด้วยหลักการแบ่งแยกและเอาชนะเช่นเดียวกับการเรียงแบบผสาน แต่อาศัยการเลือกข้อมูลมาหนึ่งตัว เรียกว่า pivot เพื่อใช้เป็นตัวแบ่งส่วนของข้อมูลออกเป็น 2 ส่วน คือ ส่วนที่น้อยกว่า pivot และส่วนที่มากกว่าหรือเท่ากับค่า pivot

ขั้นตอนการทำงานหลักของการเรียงลำดับแบบเร็ว มีดังนี้

1) เลือกข้อมูลตัวหนึ่งที่ทำหน้าที่เป็น pivot ดังรูปที่ 9.4 (ก) กำหนดให้ค่า 45 เป็นค่า pivot
2) แบ่งข้อมูลเป็น 2 ส่วนย่อย คือ ส่วนของข้อมูลที่มีค่าน้อยกว่า pivot และส่วนของข้อมูลที่มีค่ามากกว่าหรือเท่ากับ pivot โดยใช้ค่า pivot เป็นเงื่อนไขในการแบ่ง ดังรูปที่ 9.4 แสดงตัวอย่างข้อมูลจำนวน 13 ค่า เมื่อเลือกค่า pivot คือ 45 ดังรูปที่ 9.4 (ก) จะนำข้อมูลมาเปรียบเทียบกับค่า 45 ข้อมูลจะถูกแบ่งเป็น 2 ส่วน โดยข้อมูลฝั่งซ้าย คือ ข้อมูลที่น้อยกว่า 45 และข้อมูลฝั่งขวา คือ ข้อมูลที่มากกว่าหรือเท่ากับ 45 ดังรูปที่ 9.4 (ข) จะพบว่าค่า 45 จะอยู่ในลำดับที่ถูกต้องแล้ว
3) วนทำซ้ำข้อ 1) และข้อ 2) กับสองส่วนย่อยในลักษณะเดิมด้วยหลักการเรียกซ้ำ จนเหลือข้อมูลเพียงตัวเดียว



(ก) กำหนดค่า pivot **(ข) แบ่งเป็น 2 ส่วน ด้วยค่า pivot**

รูปที่ 9.4 ตัวอย่างการแบ่งข้อมูลเป็น 2 ส่วนด้วยค่า pivot

ตัวอย่างอัลกอริทึมเรียงลำดับแบบเร็ว

```
void Qsort(int[] A, int L, int R) {
    if (L < R) {
  (1) int p  <-  partition(A, L, R);
  (2) Qsort(A, L, p - 1);
  (3) Qsort(A, p + 1, R);
    }
}
```

กรณีมีข้อมูล 2 ค่าขึ้นไป

(1) แบ่งข้อมูลเป็น 2 ส่วนโดยใช้ตำแหน่ง p (ตำแหน่งของค่า pivot ในอาร์เรย์) เป็นจุดแบ่ง

(2) นำกลุ่มข้อมูลที่น้อยกว่า pivot ไปเรียงลำดับต่อด้วยหลักการเรียกซ้ำ

(3) นำกลุ่มข้อมูลที่มากกว่าหรือเท่ากับ pivot ไปเรียงลำดับต่อด้วยหลักการเรียกซ้ำ

วิธีการแบ่งข้อมูลออกเป็น 2 ส่วนด้วยค่า pivot มีรายละเอียดดังนี้

1) กำหนดชื่อตัวแปรและสัญลักษณ์ต่าง ๆ ดังนี้

- A[L ...R] คือ อาร์เรย์สำหรับจัดเก็บข้อมูล เมื่อ L คือ ตัวชี้ตำแหน่งข้อมูลตัวแรกในอาร์เรย์ และ R คือ ตัวชี้ไปยังตำแหน่งข้อมูลสุดท้ายในอาร์เรย์
- p คือ ตัวชี้ไปยังตำแหน่งข้อมูลที่กำหนดเป็นค่า pivot และ x คือ ค่า pivot (x = A[p])
- S1 คือ ส่วนข้อมูลในอาร์เรย์ที่มีค่าน้อยกว่า p เมื่อ i คือ ตัวชี้ตำแหน่งข้อมูลตัวสุดท้ายใน S1 ดังนั้น A[p+1..i] $\in$ S1
- S2 คือ ส่วนข้อมูลในอาร์เรย์ที่มีค่ามากกว่าเท่ากับ p (อยู่ถัดจากกลุ่ม S1)
- ? คือ ส่วนที่ยังไม่ได้จัดกลุ่ม (อยู่ถัดจากกลุ่ม S2) เมื่อ j คือ ตำแหน่งข้อมูลตัวแรกในส่วน ? ดังนั้น A[i+1..j-1] $\in$ S2 และ A[j .. R] $\in$?

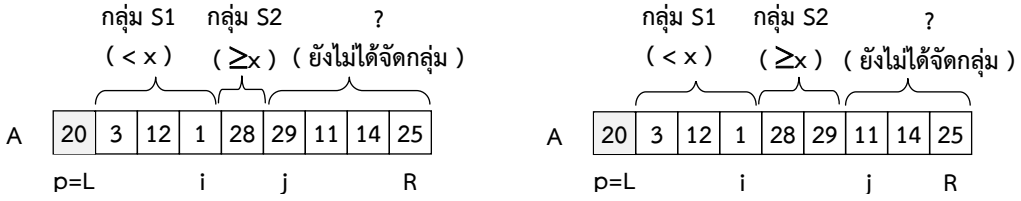
2) หลักเกณฑ์ในการจัดกลุ่ม

ข้อมูลแบ่งเป็น 2 ส่วน คือ A[p+1..i] $\in$ S1 เป็นกลุ่มข้อมูลที่น้อยกว่าค่า pivot (ค่า 20), A[i+1..j-1] $\in$ S2 เป็นกลุ่มข้อมูลที่มากกว่าหรือเท่ากับค่า pivot และส่วน A[j .. R] $\in$? เป็นกลุ่มข้อมูลที่ยังไม่ได้พิจารณาแยกกลุ่มรูปที่ 9.5 - 9.6 หลักการจัดกลุ่มให้ข้อมูแยกเป็น 2 กรณี ดังนี้

2.1) จากรูปที่ 9.5 (ก) พบว่า 29 > 20 นั่นคือ A[j] > A[p] จะจัดให้ข้อมูล A[j] อยู่กลุ่ม S2 ด้วยการขยายขนาดของกลุ่ม S2 ดังนั้นค่า j จะเพิ่มขึ้นหนึ่งค่า และค่า j จะขยับไปชี้ตำแหน่งข้อมูลตัวถัดไปที่ยังไม่เรียงในกลุ่ม ? ดังรูปที่ 9.5 (ข)

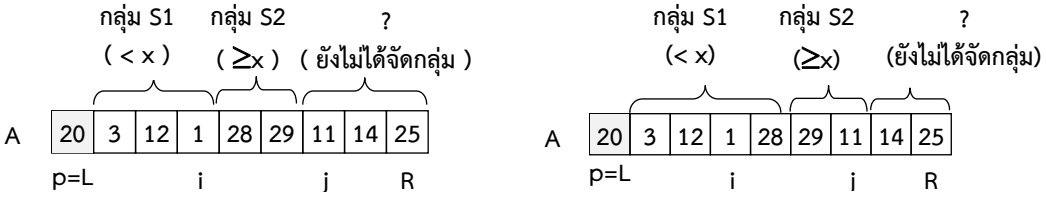
2.2) จากรูปที่ 9.6 (ก) พบว่า 11 < 20 นั่นคือ A[j] < A[p] จะจัดให้ข้อมูล A[j] มาใส่ในกลุ่ม S1 โดยขยายขนาดของกลุ่ม S1 ดังนั้นจะเพิ่มค่า i หนึ่งค่า และสลับค่าระหว่าง A[i] กับ A[j]

ด้วยคำสั่ง swap(A[i], A[j])



(ก) ก่อนการพิจารณาจัดกลุ่มข้อมูล 29 **(ข) หลังพิจารณาจัดกลุ่มข้อมูล 29 แล้ว**

รูปที่ 9.5 ตัวอย่างการพิจารณาจัดกลุ่มข้อมูล 29



(ก) ก่อนการพิจารณาจัดกลุ่มข้อมูล 11 **(ข) หลังพิจารณาจัดกลุ่มข้อมูล 11 แล้ว**

รูปที่ 9.6 ตัวอย่างการพิจารณาจัดกลุ่มข้อมูล 11

3) ขั้นตอนในการแบ่งข้อมูลออกเป็น 2 ส่วนย่อย มีดังนี้

3.1) กำหนดค่าข้อมูลที่เป็น pivot ในหัวข้อนี้ใช้ข้อมูลตัวแรกในอาร์เรย์ (กำหนด p=1)

3.2) พิจารณาข้อมูล A[j] ที่ยังไม่ได้จัดกลุ่ม, i=L+1...R เพื่อจัดให้อยู่ในกลุ่ม S1 หรือ S2 ให้ครบทุกค่า ดังรูปที่ 9.7 (ก)

3.3) ย้ายค่า A[p] ไปอยู่ในตำแหน่งที่เหมาะสม โดยจะสลับค่า A[i] กับ A[p] พบว่าตำแหน่ง i คือ ตำแหน่งที่ข้อมูลเรียงลำดับเสร็จแล้วจะคืนค่า i เป็นตำแหน่งที่ใช้ในการแบ่งข้อมูลเป็น 2 ส่วนย่อย ดังรูปที่ 9.7 (ข)

(ก) พิจารณาจัดกลุ่มข้อมูลให้ครบทุกค่า **(ข) ย้ายค่า pivot กับข้อมูลในตำแหน่ง i**

รูปที่ 9.7 หลักการแบ่งข้อมูลเป็น 2 ส่วนด้วยค่า pivot

ตัวอย่างอัลกอริทึม Partition() เป็นขั้นตอนวิธีสำหรับการแบ่งข้อมูลเป็น 2 ส่วน มีคำสั่งดังนี้

```
int Partition(int A[], L, R) //A[L. . R]
    p  <-  L, x  <- A[p]
    i  <-  L
    for j = L+1 to R
        if A[j] < x then
            i <- i+ 1
            swap(A[i], A[j])
   swap(A[p], A[i])
return(i)
```

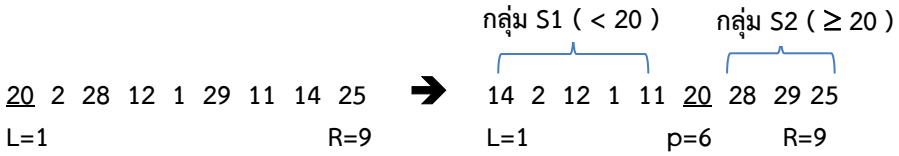
ตารางที่ 9. 5 แสดงตัวอย่างการจัดแบ่งข้อมูลเป็น 2 ส่วนย่อยด้วยอัลกอริทึม Partition() ซึ่งเริ่มต้นด้วยการใช้ค่า 20 เป็น pivot และเมื่อพิจารณาครบทุกค่า จะจัดให้ข้อมูล 3 12 1 11 และ 14 อยู่ในกลุ่ม S1 คือ เป็นส่วนข้อมูลของข้อมูลที่มีค่าน้อยกว่า 20 และจัดให้ข้อมูล 28 29 และ 25 อยู่ในกลุ่ม S2 คือ เป็นส่วนข้อมูลของข้อมูลที่มีค่ามากกว่าหรือเท่ากับ 20 โดยหลังทำงานเสร็จตำแหน่ง i จะเป็นตำแหน่งที่ใช้ในการแบ่งข้อมูลเป็น 2 ส่วนย่อย ดังแสดงในแถวสุดท้ายของตารางที่ 9.5

ตารางที่ 9.5 ตัวอย่างลำดับการทำงานของอัลกอริทึม Partition ที่ใช้ค่าแรกในอาร์เรย์เป็น pivot

ค่า j	ข้อมูลก่อนการพิจาณา A[j]	ข้อมูลหลังการพิจาณา A[j]	คำอธิบาย
2	20 3 28 12 1 29 11 14 25 i j	20 3 28 12 1 29 11 14 25 i,j	3 < 20 จึงจัดข้อมูล 3 ให้อยู่กลุ่ม S1 โดยเพิ่มค่า i
3	20 3 28 12 1 29 11 14 25 i j	-	28 > 20 จึงจัดข้อมูล 28 ให้อยู่กลุ่ม S2
4	20 3 28 12 1 29 11 14 25 i j	20 3 12 28 1 29 11 14 25 i j	12 < 20 จึงจัดข้อมูล 12 ให้อยู่กลุ่ม S1 โดยเพิ่มค่า i แล้วสลับค่า A[i] กับ A[j]
5	20 3 12 28 1 29 11 14 25 i j	20 3 12 1 28 29 11 14 25 i j	1 < 20 จึงจัดข้อมูล 1 ให้อยู่กลุ่ม S1 โดยเพิ่มค่า i แล้วสลับค่า A[i] กับ A[j]
6	20 3 12 1 28 29 11 14 25 i j	-	29 > 20 จึงจัดข้อมูล 29 ให้อยู่กลุ่ม S2
7	20 3 12 1 28 29 11 14 25 i j	20 3 12 1 11 29 28 14 25 i j	11 < 20 จึงจัดข้อมูล 11 ให้อยู่กลุ่ม S1 โดยเพิ่มค่า i แล้วสลับค่า A[i] กับ A[j]
8	20 3 12 1 11 29 28 14 25 i j	20 3 12 1 11 14 28 29 25 i j	14 < 20 จึงจัดข้อมูล 14 ให้อยู่กลุ่ม S1 โดยเพิ่มค่า i แล้วสลับค่า A[i] กับ A[j]
9	20 3 12 1 11 14 28 29 25 i j	-	25 > 20 จึงจัดข้อมูล 25 ให้อยู่กลุ่ม S2
10	14 3 12 1 11 20 28 29 25 i j	-	จัดกลุ่มครบทุกค่าแล้ว จึงสลับค่า A[p] กับ A[j] (20 กับ 14) พบว่าตำแหน่ง i คือจุดแบ่งข้อมูลออกเป็น 2 ส่วน

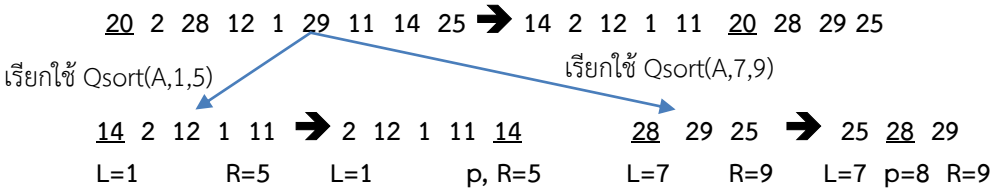
จากอัลกอริทึม Qsort() ที่กำหนดให้ข้างต้น สามารถอธิบายลำดับการทำงานได้จากตัวอย่างข้อมูลที่จัดเก็บในอาร์เรย์ A จำนวน 9 ค่า คือ 20 2 28 12 1 29 11 14 25 ได้ดังนี้

1) เริ่มต้น เมื่อเรียกใช้ Qsort(A, 1, 9) เพื่อเรียงลำดับข้อมูลในอาร์เรย์ พบว่า
   - อาร์เรย์ A มีข้อมูล 2 ค่าขึ้นไป โดยค่า L เท่ากับ 1 และ R เท่ากับ 9
   - เมื่อเรียกใช้อัลกอริทึม Partition จะแบ่งข้อมูลเป็น 2 ส่วนด้วยการใช้ค่า pivot = 20 เป็นจุดแบ่ง จะได้ผลลัพธ์ดังนี้



หลังเรียกใช้งาน Partiton() ข้อมูล 20 จะถูกจัดเรียงแล้ว จำนวน 1 ค่า และมีการเรียกใช้ Qsort(A, 1, 5) สำหรับข้อมูลฝั่งซ้าย (กลุ่ม S1) ที่น้อยกว่า 20 และ Qsort(A,7,9) และสำหรับข้อมูลฝั่งขวา (กลุ่ม S2) ที่มากกว่าหรือเท่ากับ 20

2) เนื่องจากกลุ่ม S1 มีจำนวนข้อมูล 5 ค่า จึงเรียกใช้อัลกอริทึม Partition เพื่อแบ่งข้อมูลเป็น 2 ส่วนด้วยการใช้ค่า pivot = 14 เป็นจุดแบ่ง และกลุ่ม S2 มีจำนวนข้อมูล 3 ค่า จึงเรียกใช้อัลกอริทึม Partition เพื่อแบ่งข้อมูลเป็น 2 ส่วนด้วยการใช้ค่า pivot = 28 เป็นจุดแบ่ง จะได้ผลลัพธ์ดังนี้

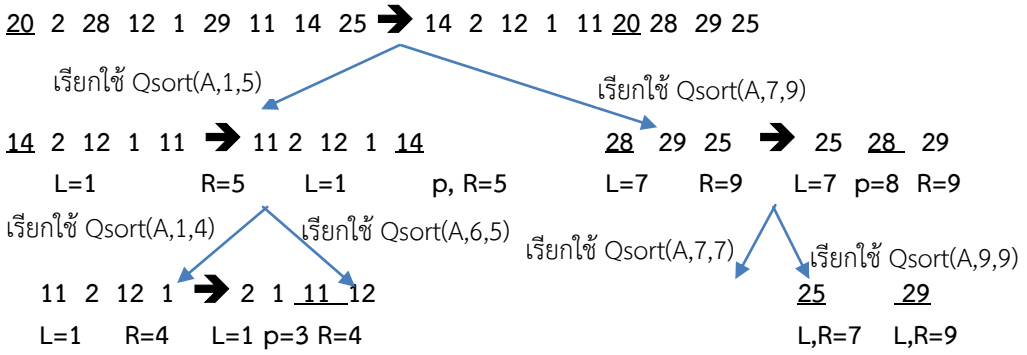


จากผลลัพธ์ที่ได้ พบว่ามีข้อมูลถูกจัดเรียงลำดับเพิ่มขึ้นอีก 2 ค่า คือ 14 และ 28 (รวม 3 ค่า คือ 14 20 28) มีรายละเอียดดังนี้

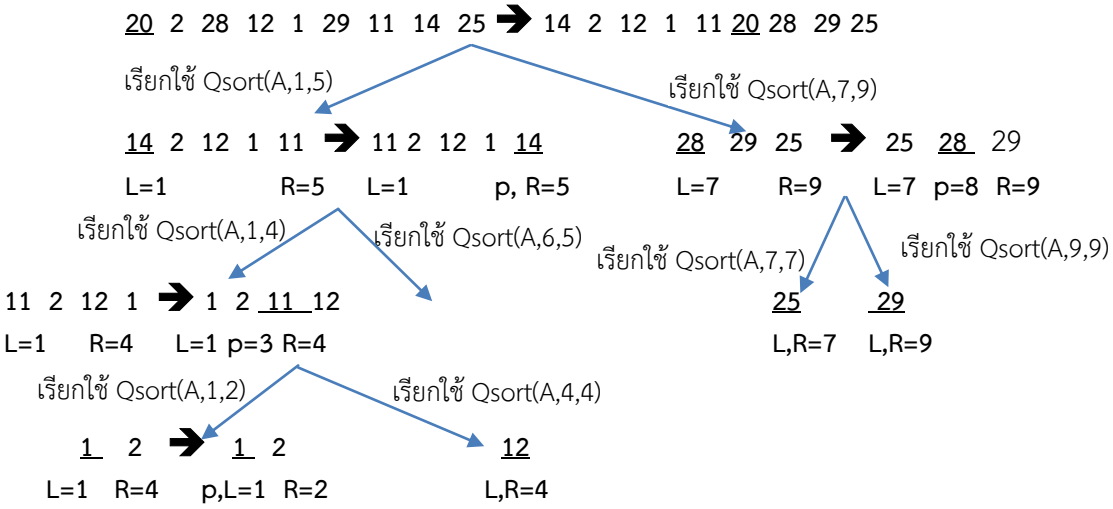
- จากกลุ่มข้อมูลฝั่งซ้ายที่น้อยกว่า 20 คือ 14 2 12 1 11 พบว่าหลังเรียกใช้งาน Partiton() ข้อมูล 14 จะถูกจัดเรียงแล้ว และ
  - มีการเรียกใช้ Qsort(A,1,4) สำหรับกลุ่มข้อมูลที่น้อยกว่า 14
  - มีการเรียกใช้ Qsort(A,6,5) สำหรับกลุ่มข้อมูลที่มากกว่าหรือเท่ากับ 14
- จากกลุ่มข้อมูลฝั่งขวาที่มากกว่าหรือเท่ากับ 20 คือ 28 29 25 พบว่าหลังเรียกใช้งาน Partiton() ข้อมูล 28 จะถูกจัดเรียงแล้ว และ
  - มีการเรียกใช้ Qsort(A,7,7) สำหรับกลุ่มข้อมูลที่น้อยกว่า 28
  - มีการเรียกใช้ Qsort(A,9,9) สำหรับกลุ่มข้อมูลที่มากกว่าหรือเท่ากับ 28

3) จากผลลัพธ์ข้างต้น จาก 4 กลุ่มข้อมูล พบว่าเมื่อเรียกใช้งาน Qsort() จะมีเพียงกลุ่มแรกของการเรียกใช้ Qsort(A,1,4) ที่มีจำนวนข้อมูลตั้งแต่ 2 ค่าขึ้นไปที่จะถูกนำไปเรียงลำดับต่อ จึงมีการเรียกใช้อัลกอริทึม Partition เพื่อแบ่งข้อมูลเป็น 2 ส่วนด้วยการใช้ค่า pivot = 11 เป็นจุดแบ่ง ผลลัพธ์ที่ได้ พบว่ากลุ่มข้อมูลฝั่งซ้ายที่น้อยกว่า 14 คือ 11 2 12 1 ที่ถูกเรียงลำดับต่อ ซึ่งหลังเรียกใช้งาน Partiton() ข้อมูล 11 จะถูกจัดเรียงแล้ว โดยที่จะมีการเรียกใช้ Qsort(A,1,2) สำหรับกลุ่มข้อมูลที่น้อยกว่า 11 และมีการเรียกใช้ Qsort(A,4,4) สำหรับกลุ่มข้อมูลที่มากกว่าหรือเท่ากับ 11 เพื่อเรียงลำดับต่อไป

โดยในขั้นตอนนี้ จะมีข้อมูลถูกจัดเรียงลำดับเพิ่มขึ้นอีก 3 ค่า คือ 11 25 และ 29 รวมทั้งหมดจำนวนทั้งหมด 6 ค่า คือ 11 14 20 25 28 29 ดังนี้



4) จากการเรียกใช้งาน Qsort(A,1,2) และ Qsort(A,4,4) ในขั้นตอนก่อนหน้านี้ จะมีเพียงกลุ่มแรกที่มีจำนวนข้อมูลตั้งแต่ 2 ค่าขึ้นไปที่จะถูกนำไปเรียงลำดับต่อด้วยการเรียกใช้อัลกอริทึม Partition เพื่อแบ่งข้อมูลเป็น 2 ส่วนด้วยการใช้ค่า pivot = 1 เป็นจุดแบ่ง ได้ผลลัพธ์ดังนี้

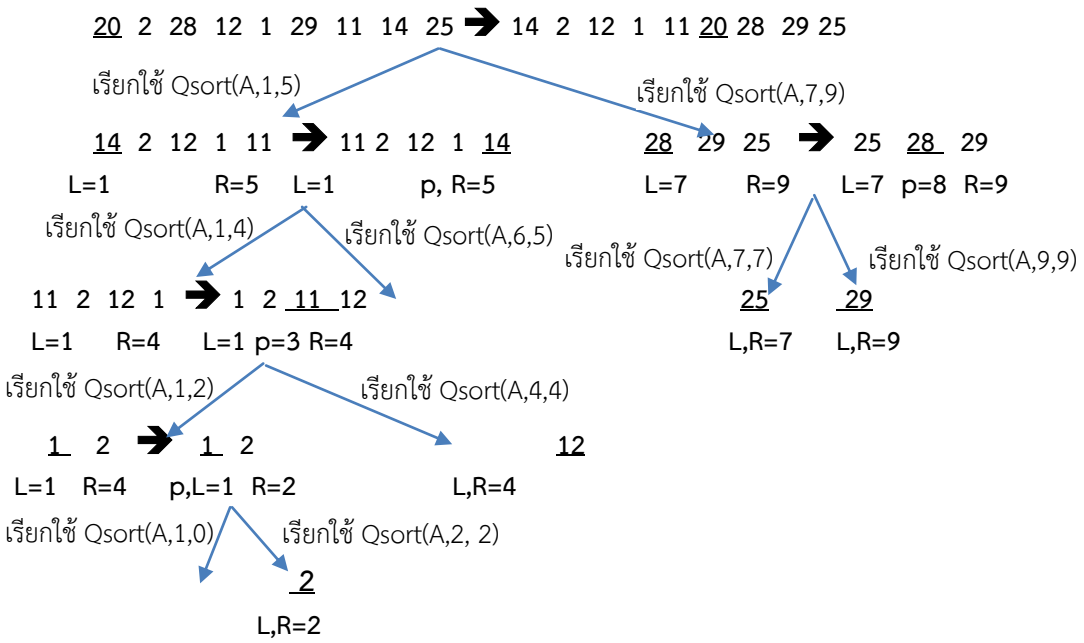


จากผลลัพธ์ข้างต้น พบว่ามีเพียงกลุ่มข้อมูลฝั่งซ้ายที่น้อยกว่า 1 1 คือ 1 2 ที่ถูกเรียงลำดับต่อ ซึ่งหลังเรียกใช้งาน Partiton() ข้อมูล 1 จะถูกจัดเรียงแล้ว โดยที่จะมีการเรียกใช้ Qsort(A,1,0) สำหรับกลุ่มข้อมูลที่น้อยกว่า 1 และมีการเรียกใช้ Qsort(A,2,2) สำหรับกลุ่มข้อมูลที่มากกว่าหรือเท่ากับ 1 เพื่อเรียงลำดับต่อไป

โดยในขั้นตอนนี้ จะมีข้อมูลถูกจัดเรียงลำดับเพิ่มขึ้นอีก 2 ค่า คือ 1 และ 12 รวมทั้งหมดจำนวน 8 ค่า คือ 1 11 12 14 20 25 28 29

5) จากการเรียกใช้งาน Qsort(A,1,0) และ Qsort(A,2,2) ในขั้นตอนก่อนหน้านี้ พบว่าไม่มีกลุ่มที่มีจำนวนข้อมูลตั้งแต่ 2 ค่าขึ้นไปอีก จึงสิ้นสุดการเรียกใช้งาน Qsort()

ในขั้นตอนสุดท้ายนี้ จะมีข้อมูลที่ถูกจัดเรียงลำดับเพิ่มอีกหนึ่งค่า คือ 2 และสิ้นสุดกระบวนการทำงานของอัลกอริทึมเรียงลำดับแบบเร็ว ดังนั้นจึงเป็นการเรียงลำดับข้อมูลรวมทั้งหมดจำนวน 9 ค่า คือ 1 2 11 12 14 20 25 28 29 ดังนี้


20 2 28 12 1 29 11 14 25 14 2 12 1 11 20 28 29 25
เรียกใช้ Qsort(A,1,5)
เรียกใช้ Qsort(A,7,9)
14 2 12 1 11 11 2 12 1 14 28 29 25 25 28 29
L=1 R=5 L=1 p, R=5 L=7 R=9 L=7 p=8 R=9
เรียกใช้ Qsort(A,1,4)
เรียกใช้ Qsort(A,6,5)
เรียกใช้ Qsort(A,7,7)
เรียกใช้ Qsort(A,9,9)
11 2 12 1 1 2 11 12 25 29
L=1 R=4 L=1 p=3 R=4 L,R=7 L,R=9
เรียกใช้ Qsort(A,1,2)
เรียกใช้ Qsort(A,4,4)
1 2 1 2 12
L=1 R=4 p,L=1 R=2 L,R=4
เรียกใช้ Qsort(A,1,0)
เรียกใช้ Qsort(A,2, 2)
2
L,R=2

แบบฝึกหัด

1. กำหนดข้อมูลในอาร์เรย์ คือ 8 6 2 5 7 4 1 9 3 จงแสดงลำดับการเรียงตัวเลขจากน้อยไปมาก ตามขั้นตอนวิธีดังต่อไปนี้
   1.1 การเรียงลำดับแบบฟอง
   1.2 การเรียงลำดับแบบแทรก
   1.3 การเรียงลำดับแบบผสาน
2. การเรียงลำดับแบบเร็วสามารถใช้วิธีการแบ่งข้อมูลออกเป็น 2 กลุ่ม กลุ่มแรก คือ กลุ่มที่มีค่าน้อยกว่าหรือเท่ากับ pivot และกลุ่มที่สอง คือ กลุ่มที่มีค่ามากกว่า pivot จงยกตัวอย่างประกอบการแสดงลำดับการเรียงลำดับด้วยวิธีการเรียงลำดับแบบเร็วด้วยการใช้ข้อมูลในตำแหน่งตรงกลางของแถวลำดับเป็นค่า pivot
3. จงเขียนโปรแกรมเพื่อเรียงลำดับอักขระจากน้อยไปมาก โดยใช้วิธีการเรียงลำดับแบบฟอง
4. จงเขียนโปรแกรมเพื่อเรียงลำดับข้อความจากมากไปน้อย โดยใช้วิธีการเรียงลำดับแบบเลือก
5. จงเขียนโปรแกรมเพื่อเรียงลำดับจำนวนจริงจากมากไปน้อย โดยใช้วิธีการเรียงลำดับแบบผสาน
6. จงเขียนโปรแกรมเพื่อเรียงคำศัพท์ในอาร์เรย์ขนาด N x N มิติ จากน้อยไปมาก