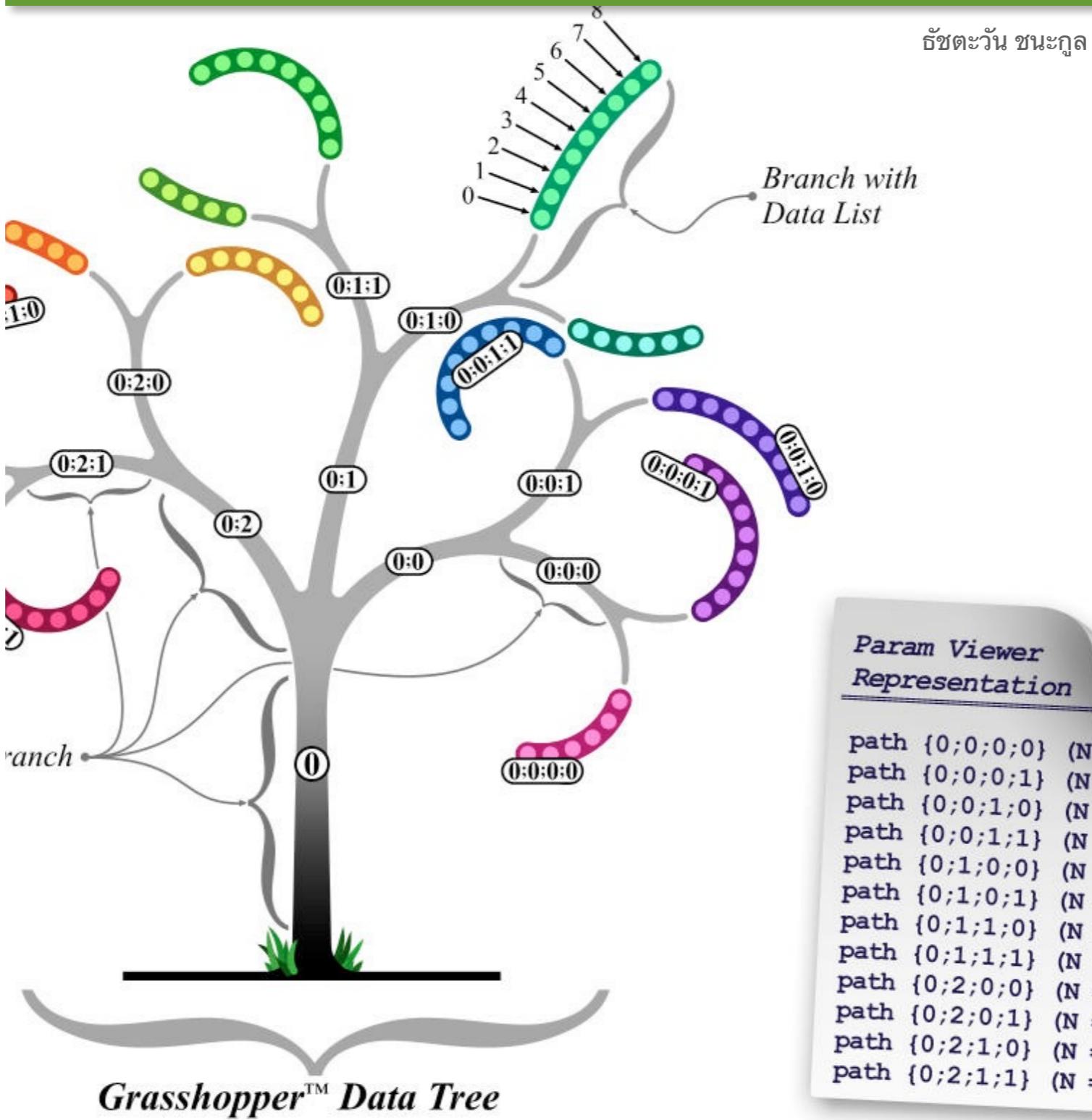


204-201

Data Structure and Algorithm



Intro-Data Structure and Algorithm

จะรู้ได้อย่างไรว่า โปรแกรมที่เขียน มีประสิทธิภาพเป็นอย่างไร?

ก่อนอื่นขอให้มาดูตัวอย่างง่าย ๆ เกี่ยวกับการกำหนดค่า ให้กับตัวแปรดังตัวอย่างต่อไปนี้ โดยให้เครื่องหมาย = หมายถึงการกำหนดค่า และก่อนเริ่มทำงาน ให้ค่า

$x = 5$ มีความหมายคือ ให้ตัวแปร x เก็บค่า 5

$y = 2$ มีความหมายคือ ให้ตัวแปร y เก็บค่า 2

Program 1: ต้องการ слับค่า x และ y โดยให้ x เก็บค่า 2 และ y เก็บค่า 5

ขั้นตอนที่ 1: $x = y$

ขั้นตอนที่ 2: $y = x$

ขั้นตอนที่ 3: stop

ขั้นตอนการทำงาน	ค่าของ x	ค่าของ y
ก่อนขั้นตอนที่ 1	5	2
หลังขั้นตอนที่ 1	2	2
หลังขั้นตอนที่ 2	2	2

โปรแกรมนี้ ต้องการที่จะสลับค่า
แต่ในความเป็นจริง ผลลัพธ์ผิด



ต้องแก้อย่างไร



Program 2:

วิธีคิด : หากฝากรหัสข้อมูลเก่า เก็บไว้ก่อน

ขั้นตอนที่ 1: $xold = x$

ขั้นตอนที่ 2: $yold = y$

ขั้นตอนที่ 3: $y = xold$

ขั้นตอนที่ 4: $x = yold$

ขั้นตอนที่ 5: stop

ต่างจากโปรแกรมแรก 2
อย่าง คือ

1. ขั้นตอนการทำงานจาก 2
เป็น 4 (ไม่รวม Stop)

2. การใช้เนื้อที่หน่วยความ
จำ

Program 3:

วิธีคิด : ถ้าไม่ใช้หน่วยความจำเพิ่มเลยล่ะ จะเป็นไปได้ไหม

ขั้นตอนที่ 1: $x = x + y$

ขั้นตอนที่ 2: $y = y - x$

ขั้นตอนที่ 3: $x = x + y$

ขั้นตอนที่ 4: $y = -y$

ขั้นตอนที่ 5: stop

ขั้นตอนการทำงาน	ค่าของ x	ค่าของ y
ก่อนขั้นตอนที่ 1	5	2
หลังขั้นตอนที่ 1	7	2
หลังขั้นตอนที่ 2	7	-5
หลังขั้นตอนที่ 3	2	-5
หลังขั้นตอนที่ 4	2	5

จำนวนครั้ง VS การใช้หน่วยความจำ

ถ้าเปรียบเทียบความสำคัญของจำนวนขั้นตอนเทียบกับเนื้อที่ในหน่วยความจำที่ใช้แล้ว จำนวนขั้นตอนที่น้อยจะมีความสำคัญกว่าการใช้หน่วยความจำ เพราะการมีขั้นตอนน้อยจะทำให้การทำงานของโปรแกรมเร็วขึ้น

Program: 2 แก้ไขให้เป็น Program: 4

ขั้นตอนที่ 1: $z = x$

ขั้นตอนที่ 2: $x = y$

ขั้นตอนที่ 3: $y = z$

ขั้นตอนที่ 4: stop

สรุปได้ว่า โปรแกรมที่ 4 ดีกว่าโปรแกรมที่ 2 ทั้งด้านความเร็ว และการใช้ตัวแปรในหน่วยความจำ

ส่วนโปรแกรมที่ 3 จะดีกว่าโปรแกรมที่ 2 ในด้านการใช้เนื้อที่ในหน่วยความจำ

โครงสร้างข้อมูล (Data Structure)

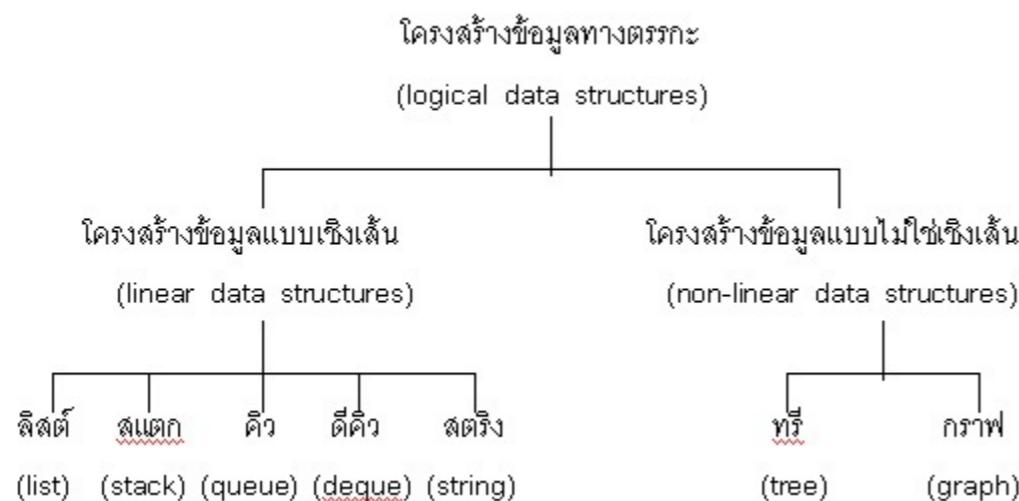
หน่วยข้อมูลย่อยๆ ที่ถูกจัดวางในรูปแบบที่เหมาะสมแล้ว กำหนดลักษณะความสัมพันธ์และความเชื่อมโยงทางตรรกะเพื่อ การนำมาประยุกต์ใช้งานในโปรแกรม

การรวมประเภทข้อมูลเข้าไว้ด้วยกันจนกระทั่งภายใน เป็นกลุ่มของ ประเภทข้อมูล และมีการกำหนดค่านิยามของความสัมพันธ์ภายใน กลุ่มข้อมูลไว้อย่างชัดเจน

wiki - a way of storing data in a computer so that it can be used efficiently

โครงสร้างข้อมูลทางตรรกะ

โครงสร้างข้อมูลทางตรรกะ (Logical data structures) เป็น โครงสร้างข้อมูลที่เกิดจากจินตนาการของผู้ใช้เพื่อใช้แก้ปัญหาใน โปรแกรมที่สร้างขึ้น จำแนกได้เป็น 2 ประเภท



โครงสร้างข้อมูลแบบเชิงเส้น

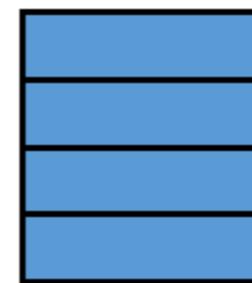
โครงสร้างข้อมูลแบบเชิงเส้น (linear data structures) เป็น ชนิดข้อมูลที่ความสัมพันธ์ของข้อมูลเรียงต่อเนื่องกัน โดยข้อมูลตัว ที่ 2 อยู่ต่อจาก ข้อมูลตัวที่ 1 ข้อมูลตัวที่ 3 อยู่ต่อจากข้อมูลตัวที่ 2 และข้อมูลตัวที่ n อยู่ต่อจากข้อมูลตัวที่ $n-1$

ตัวอย่าง โครงสร้างข้อมูลแบบเชิงเส้น เช่น

❖ ลิสต์ (list)



❖ อาร์เรย์ (array)



❖ สแตก (stack)

❖ คิว (queue)

❖ เดคิว (dequeue)

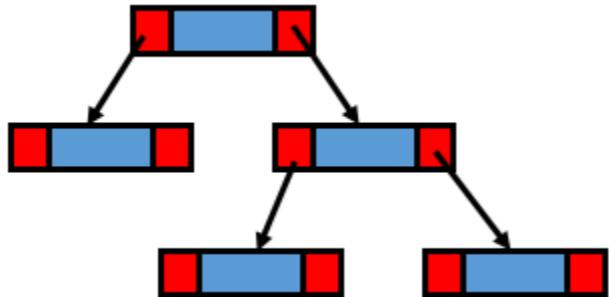
❖ สตริง (string)

โครงสร้างข้อมูลแบบไม่เชิงเส้น

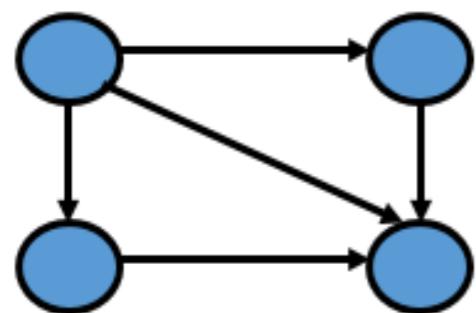
โครงสร้างข้อมูลแบบไม่ใช่เชิงเส้น (Non-linear data structures) เป็นชนิดข้อมูลที่ข้อมูลแต่ละตัวสามารถมีความสัมพันธ์กับข้อมูลอื่นได้หลายตัว

ตัวอย่าง โครงสร้างข้อมูลแบบไม่ใช่เชิงเส้น

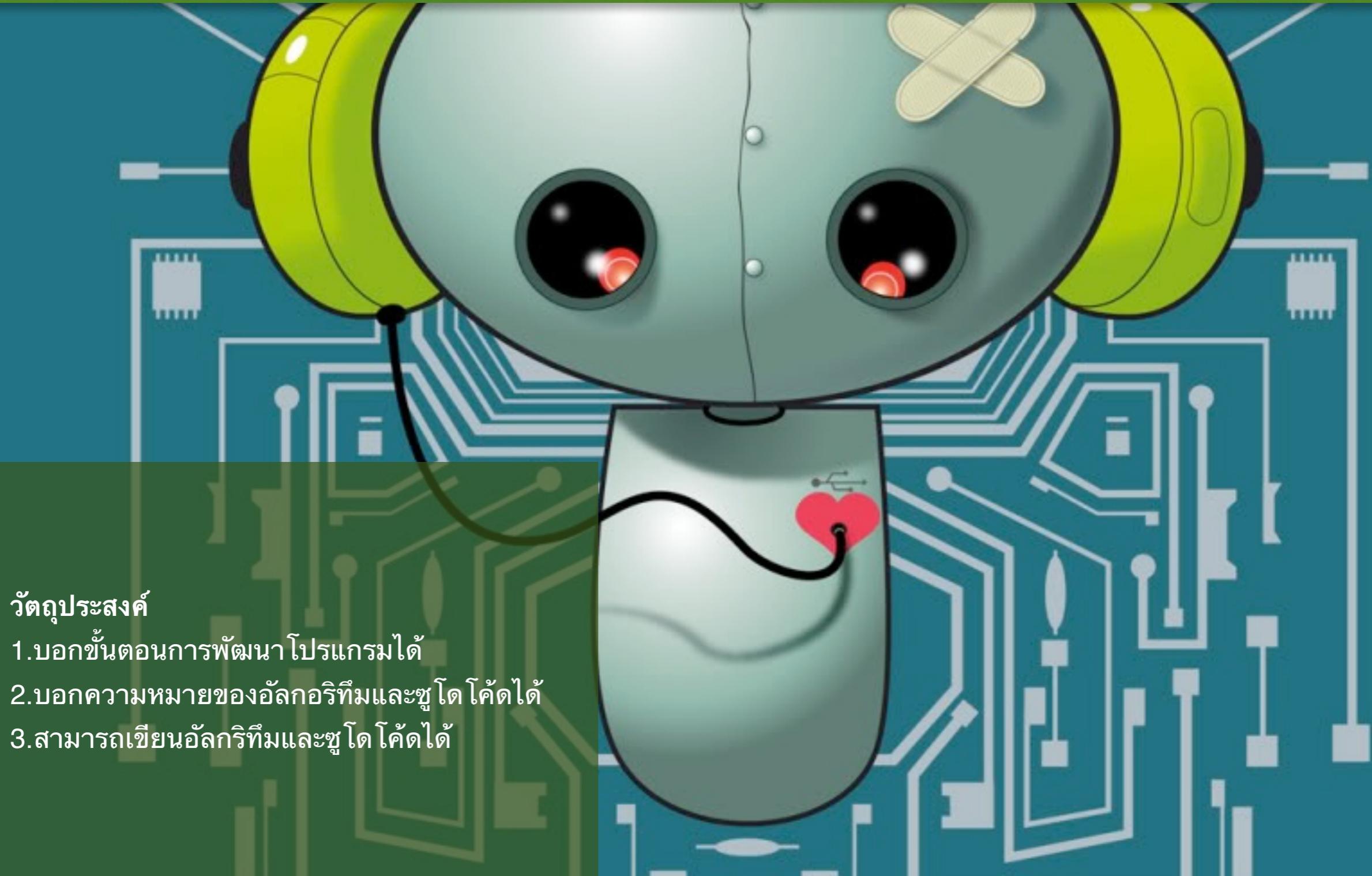
❖ ทรี (tree)



❖ กราฟ (graph)



Algorithm



วัตถุประสงค์

1. บอกขั้นตอนการพัฒนาโปรแกรมได้
2. บอกความหมายของอัลกอริทึมและชูโด โคดได้
3. สามารถเขียนอัลกอริทึมและชูโด โคดได้

Section 1

นิยามอัลกอริธึม

สิ่งสำคัญในการเขียนโปรแกรมก็คือ ความมีหลักวิธีคิดที่ดี รวมถึงโปรแกรมต้องได้รับการออกแบบอย่างมีแบบแผนก่อน ที่จะดำเนิน ในขั้นตอนการเขียน โปรแกรมต่อไป โดย กระบวนการที่ได้รับการออกแบบอย่างมีแบบแผน ในที่นี้ก็คือ อัลกอริธึม นั่นเอง

หากพูดโดยง่าย อัลกอริธึม ก็ไม่ได้แตกต่างไปจากการ ทำอาหาร ที่แต่ละคนสามารถปฏิบัติตาม ขั้นตอนการปรุงตาม แต่ละเมนูที่ตำราแนะนำ รวมถึงเครื่องปรุง อุปกรณ์ ผู้เรียน ที่ศึกษา ก็จะสามารถปรุงอาหารตามตำราและก็จะได้อาหาร ชนิดที่ต้องการทุกคน

ดังนั้น

อัลกอริธึม (**Algorithm**) หมายถึง ลำดับขั้นตอนวิธีใน การทำงานของ โปรแกรมเพื่อแก้ปัญหาปัญหานั่น ชึ่งถ้า ปฏิบัติตามขั้นตอนอย่างถูกต้องแล้ว จะต้องสามารถช่วยแก้ ปัญหาหรือประมวลผลตามความต้องการได้สำเร็จ **สำหรับ** ใน การเขียนอธิบายอัลกอริธึมนั้น เราสามารถคิดอัลกอริธึมเพื่อ มาแก้ปัญหาได้หลายแบบ



ตัวอย่างอัลกอริธึม(Algorithm) การต้มไข่ไก่

วัตถุดิบ : ไข่ไก่ ผลลัพธ์: ไข่ต้มสุก

Algorithm 1	Algorithm 2	Algorithm 2
ขั้นตอนที่ 1: ต้มน้ำให้เดือด	ขั้นตอนที่ 1: ต้มน้ำให้เดือด	ขั้นตอนที่ 1: ต้มน้ำให้เดือด
ขั้นตอนที่ 2: ใส่ไข่	ขั้นตอนที่ 2: ใส่ไข่	ขั้นตอนที่ 2: รอ 10 นาที
ขั้นตอนที่ 3: รอ 10 นาที	ขั้นตอนที่ 3: รอ 5 นาที	ขั้นตอนที่ 3: ตับไฟ / ปิดเตา
ขั้นตอนที่ 4: ตับไฟ / ปิดเตา	ขั้นตอนที่ 4: ตับไฟ / ปิดเตา	ขั้นตอนที่ 4: ปอกไข่
ขั้นตอนที่ 5: ปอกไข่		

Algorithm 1 Vs Algorithm 2

ผลที่ได้เหมือนกันคือ ไข่ต้ม

ผลลัพธ์ Algorithm 1 สามารถทำได้เลย ส่วน

Algorithm 2 ต้องปอกก่อนทาน

สรุปคือ เราได้ผลลัพธ์ตามที่โจทย์ต้องการคือ ไข่ต้ม

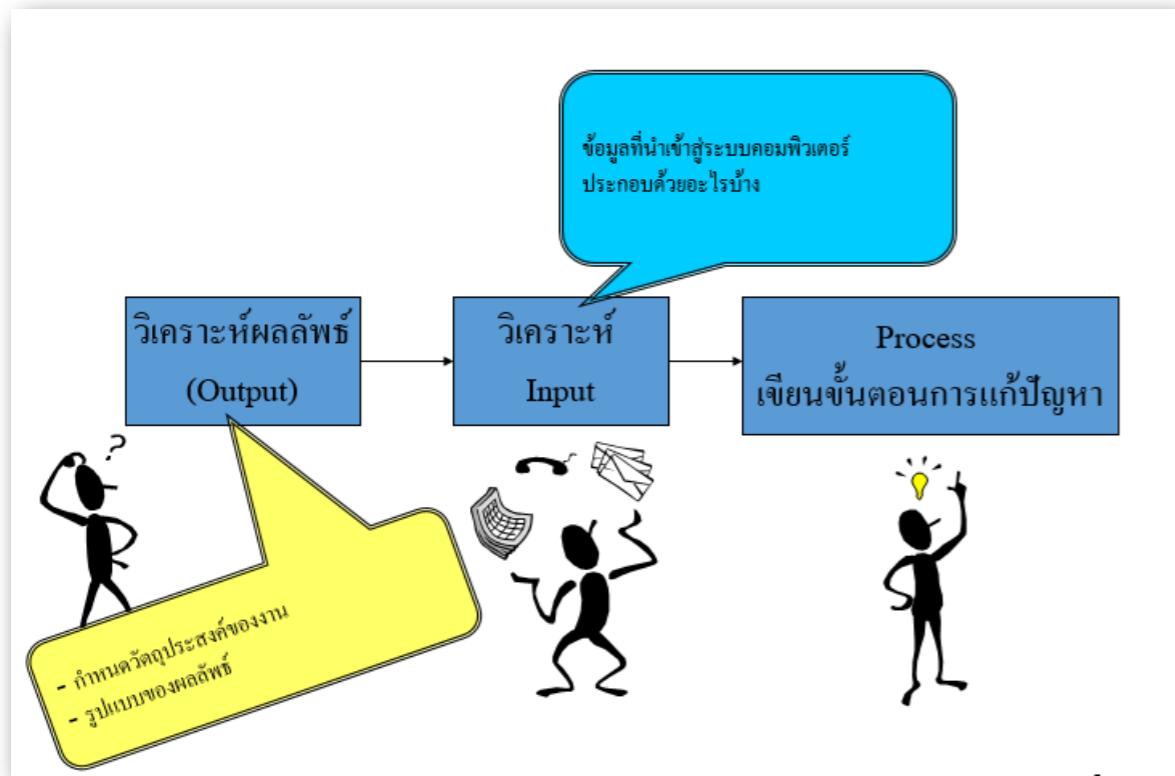
Algorithm 3

คุณยังไม่ได้ไข่ต้ม..... เพราะ

คุณยังไม่ได้ใส่ไข่ต้มลงไปน้ำเอง

Section 2

การวิเคราะห์ปัญหา



การพิจารณา

ขั้นตอนการทำงานเป็นการนำเข้าป้อนเข้าระบบถือเป็น **Input**

ขั้นตอนเกี่ยวกับการกระทำ(กริยา) ถือเป็น **Process**

ขั้นตอนการนำข้อมูลออกจากระบบ และแสดงผล ถือเป็น **Output**

เราลองนำขั้นตอนการต้มไข่มาวิเคราะห์

ต้มน้ำให้เดือด => การกระทำ(Process)

ใส่ไข่ => การป้อนข้อมูล(Input)

รอ 10 นาที => การกระทำ(Process)

ดับไฟ => การกระทำ(Process)

ปอกไข่ => การกระทำ(Process)

ผลลัพธ์ => ไข่ต้มสุก (Output)

ตัวอย่าง ต้องการคำนวณหาพื้นที่ของสามเหลี่ยมรูปหนึ่ง

1. วิเคราะห์ผลลัพธ์ : พื้นที่สามเหลี่ยม กำหนดวัตถุประสงค์ การคำนวณหาพื้นที่สามเหลี่ยม

รูปแบบผลลัพธ์(Output)

พื้นที่สามเหลี่ยม =

2. กำหนดข้อมูลเข้า (Input)

ความยาวฐาน

ความสูง

3. ขั้นตอนการประมวลผล (Process)

ขั้นที่ 1 ป้อนความยาวฐาน

ขั้นที่ 2 ป้อนความสูง

ขั้นที่ 3 คำนวณพื้นที่สามเหลี่ยม จากสูตร

$$\text{พื้นที่สามเหลี่ยม} = \frac{1}{2} \times \text{ความยาวฐาน} \times \text{ความสูง}$$

ตัวอย่าง อัลกอริธึมเพื่อทำการบวกราคาโดยใช้เครื่องคิดเลข

1. วิเคราะห์ผลลัพธ์ : ยอดรวมราคา

2. กำหนดข้อมูลเข้า(Input)

ยอดเงิน

3. ขั้นตอนการประมวลผล (Process)

ขั้นที่ 1 พิมพ์ยอดเงิน

ขั้นที่ 2 กดเครื่องหมาย

ขั้นที่ 3 วนการทำงาน

ขั้นที่ 4 กดเครื่องหมาย =

ขั้นที่ 5 คำนวณยอดรวมราคา

ตัวอย่าง โยนเหรียญเลี้ยงทายเพื่อตัดสินใจว่าจะกินขنمปังหรือผลไม้ โดยมีเงื่อนไขว่า ถ้าออกหัวกินขنمปัง ถ้าออกก้อยกินผลไม้

1. วิเคราะห์ผลลัพธ์ : กินอะไร(ผลไม้/ขنمปัง)

2. กำหนดข้อมูลเข้า(Input) : ผลการโยน

3. ขั้นตอนการประมวลผล (Process)

ขั้นที่ 1 ดูเหรียญ => รับข้อมูล

ขั้นที่ 2 ถ้าออกหัว => ไปขั้นตอนที่ 4

ขั้นที่ 3 ถ้าออกก้อย => ไปขั้นตอนที่ 6

ขั้นที่ 4 กินขنمปัง => ไปขั้นตอนที่ 6

ขั้นที่ 5 กินผลไม้ => ไปขั้นตอนที่ 6

ขั้นที่ 6 หยุด

ตัวอย่าง ต้องการหายอดรวมราคาสินค้าที่ซื้อ

1. วิเคราะห์ผลลัพธ์ : ราคาสินค้าทั้งหมด

2. กำหนดข้อมูลเข้า(Input) :

ราคาสินค้าต่อชิ้น

จำนวนสินค้าที่ซื้อ

3. ขั้นตอนการประมวลผล (Process)

ขั้นที่ 1 รับจำนวนสินค้าที่ซื้อ

ขั้นที่ 2 รับราคาสินค้าต่อชิ้น

ขั้นที่ 3 ราคาสินค้า = นำราคាត่อชิ้น * จำนวนที่ซื้อ

ขั้นที่ 4 หากมีสินค้าอีกกลับไปที่ข้อ 1

ขั้นที่ 5 นำราคасินค้าในข้อที่ 3 มาบวกกัน

Section 3

การออกแบบอัลกอริธึม

ในการเขียนอธิบายอัลกอริธึมนั้น เราสามารถคิดอัลกอริธึมเพื่อมาแก้ปัญหาได้หลายแบบ ซึ่งในแต่ละแบบเครื่องคอมพิวเตอร์จะใช้ในหน่วยความจำ และเวลาในการประมวลผลไม่เท่ากัน ดังนั้น การจะเปรียบเทียบว่าโปรแกรมคอมพิวเตอร์ควรเก่งกว่ากันนั้นจึงใช้การเปรียบเทียบและประสิทธิภาพของอัลกอริธึมนั้นเอง

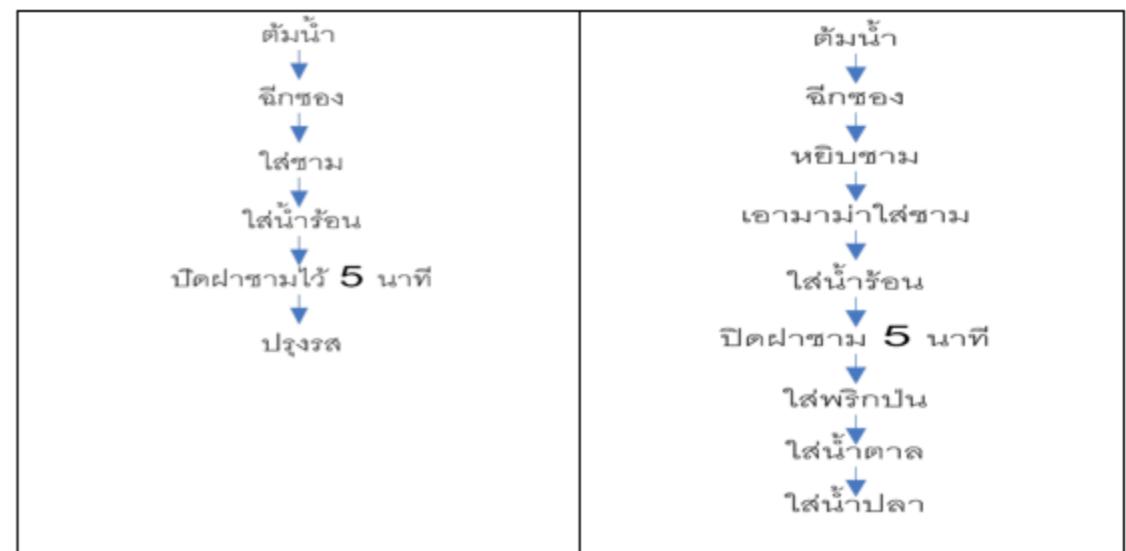
อัลกอริธึมของใครใช้เวลาในการประมวลผลและหน่วยความจำน้อยกว่า ถือว่าอัลกอริธึมนั้นadalกว่าอีกอัลกอริธึม

ประสิทธิภาพของอัลกอริธึม

จะพิจารณาอยู่ 2 ส่วนหลักๆ ดังนี้

- หน่วยความจำ(Memory) ที่จะต้องใช้ในการประมวลผล
- เวลา(Time) ที่ใช้ในการประมวลผล

Algorithm ต้มมาม่า



อัลกอริธึมที่ดีจะประกอบด้วยคุณสมบัติต่างๆ ดังนี้

- อัลกอริธึมที่ดีต้องมีความถูกต้อง (Correctness)
- อัลกอริธึมที่ดีต้องง่ายต่อการอ่าน(Readability)
- อัลกอริธึมที่ดีต้องสามารถปรับปรุงได้ง่ายต่ออนาคต(Ease of modification)
- อัลกอริธึมที่ดีสามารถนำกลับมาใช้ใหม่ได้(Reusability)
- อัลกอริธึมที่ดีต้องมีประสิทธิภาพ (Efficiency)

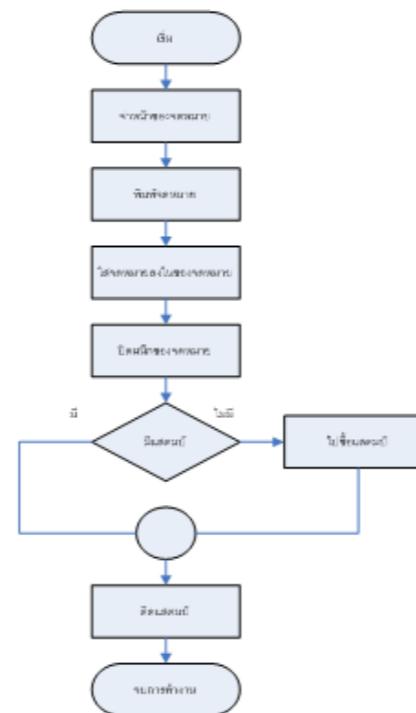
การควบคุมการทำงานของโปรแกรม (Program Control Flow)

การควบคุมการทำงานของโปรแกรม เป็นเครื่องมือที่ผู้พัฒนาโปรแกรมใช้ในการแสดงลำดับการทำงานของโปรแกรมหรือใช้อธิบายอัลกอริธึม ให้เป็นระบบและง่ายต่อความเข้าใจ โดยโครงสร้างอาจจะอยู่ในรูปแบบดังนี้คือ

1. ผังงาน (Flowchart) ซึ่งเป็น Flow Diagram ชนิดหนึ่งสำหรับใช้อธิบายขั้นตอนการทำงานของโปรแกรมในลักษณะรูปภาพ

2. รหัสเทียม (Pseudo code) จะมีสัญลักษณ์คล้ายกับภาษาอังกฤษ กำกึงระหว่างภาษาอังกฤษกับภาษาคอมพิวเตอร์ ใช้ในการอธิบายลักษณะโครงสร้างของข้อมูล และการทำงานของอัลกอริธึมที่เราเขียนขึ้น

ตัวอย่าง Flow Chart และ Pseudo Code

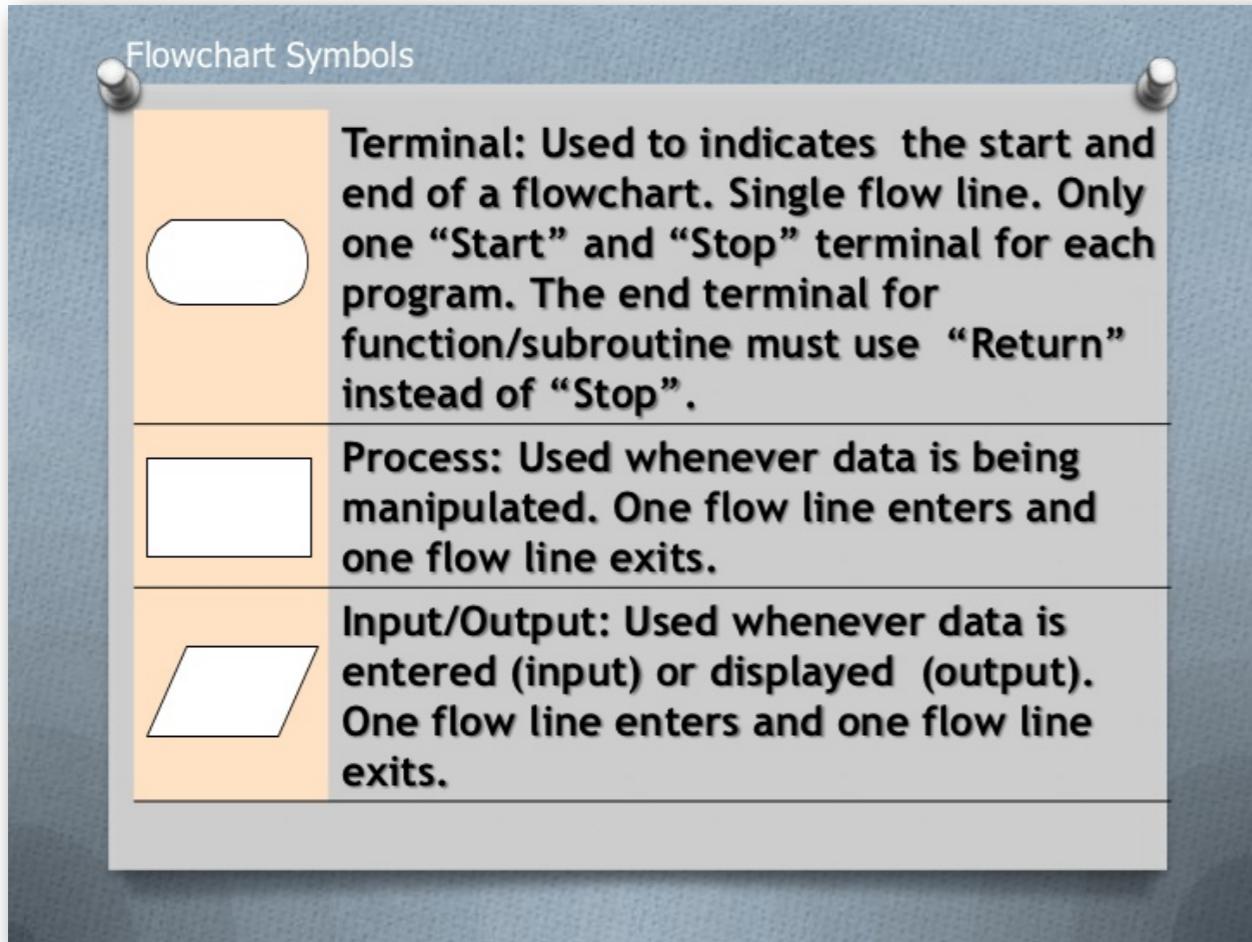


Algorithm Summation

1. $\text{Sum} = 0$
 2. $\text{Input}(\text{value1})$
 3. $\text{Input}(\text{value2})$
 4. $\text{Input}(\text{value3})$
 5. $\text{Sum} = \text{value1} + \text{value2} + \text{value3}$
 6. $\text{Output}(\text{sum})$
- End .

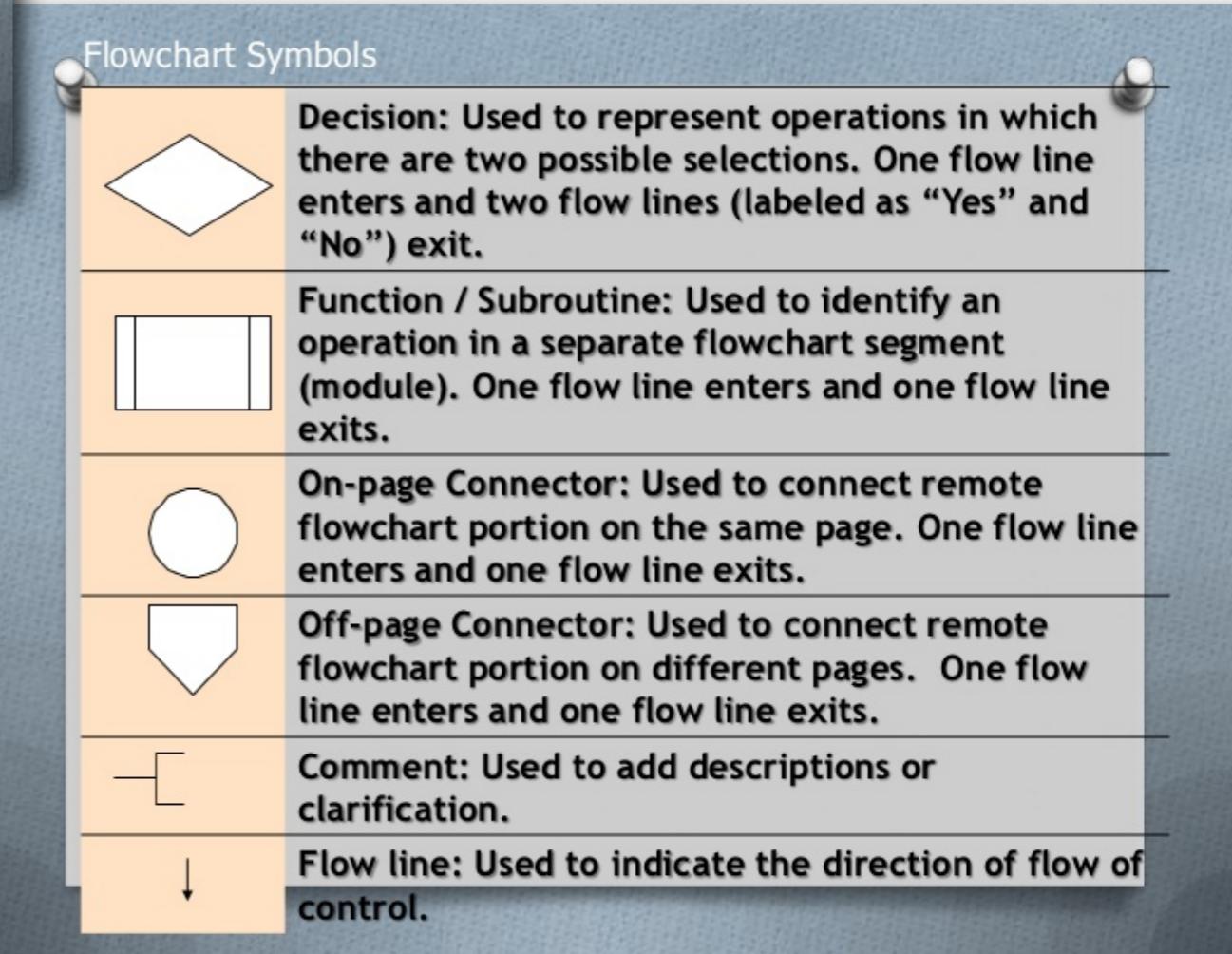
การเขียนอัลกอริธึมการบวกเลข 3 ตัว ในรูปแบบ Pseudo

สัญลักษณ์ในผังงาน (Flow Chart)

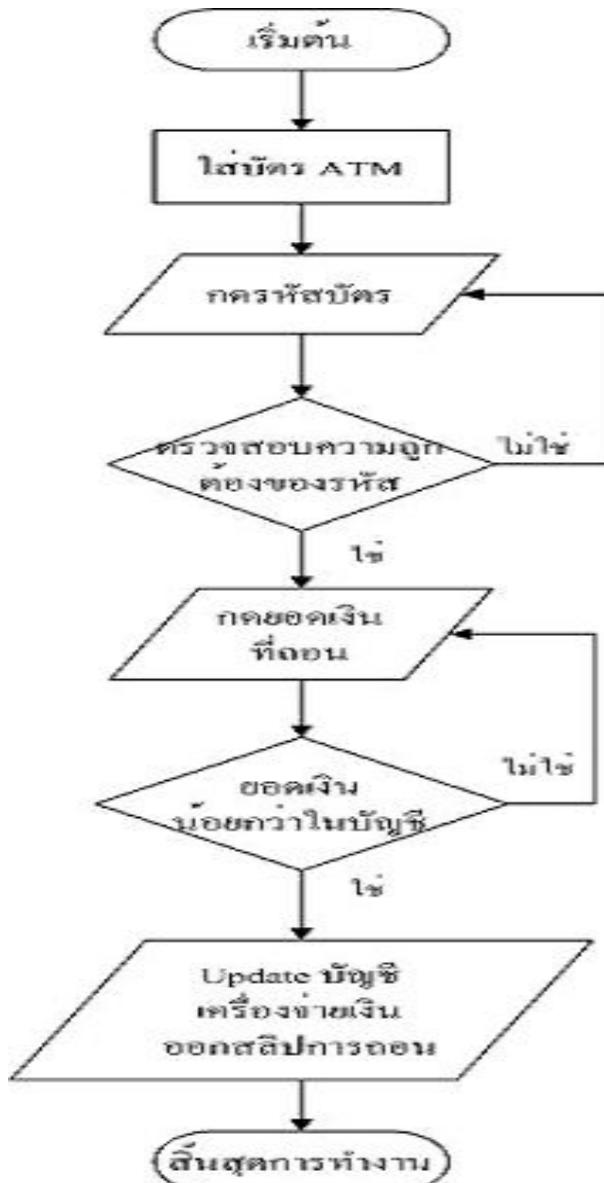


ประโยชน์ของผังงาน

1. ลำดับขั้นตอนการทำงานของโปรแกรม และสามารถนำไปเขียนโปรแกรมได้โดยไม่สับสน
2. ตรวจสอบความถูกต้อง และแก้ไขโปรแกรมได้ง่าย เมื่อเกิดข้อผิดพลาด
3. การปรับปรุง เปลี่ยนแปลง แก้ไข ทำได้อย่าง สะดวกและรวดเร็ว
4. ทำให้ผู้อื่นสามารถศึกษาการทำงานของ โปรแกรมได้อย่างง่าย และรวดเร็วมากขึ้น



ตัวอย่างผังงาน การถอนเงินจาก ATM



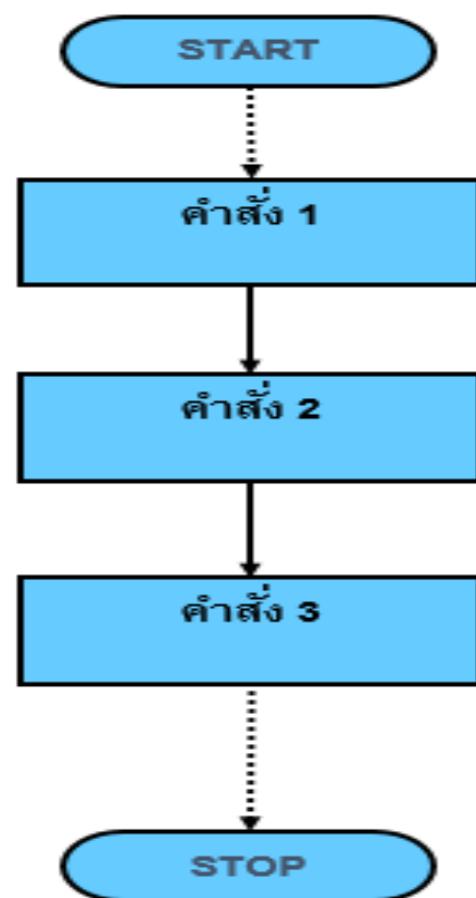
วิธีการเขียนผังงานที่ดี

1. ใช้สัญลักษณ์ตามที่กำหนดไว้
2. ใช้ลูกศรแสดงทิศทางการไหลของข้อมูลจากบนลงล่าง หรือจากซ้ายไปขวา
3. คำอธิบายในภาพสัญลักษณ์ผังงานควรสั้นกะทัดรัด และเข้าใจง่าย
4. ทุกแผนภาพต้องมีลูกศรแสดงทิศทางเข้า - ออก
5. ไม่ควรໂヨงเส้นเชื่อมผังงานที่อยู่ใกล้มาก ๆ ควรใช้สัญลักษณ์จุดเชื่อมต่อแทน
6. ผังงานควรมีการทดสอบความถูกต้องของการทำงาน ก่อนนำไปเขียนโปรแกรมจริง

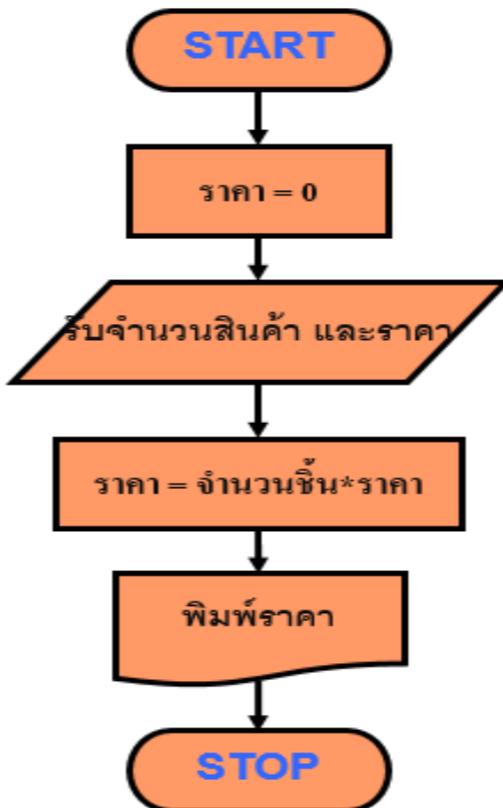
โครงสร้างพื้นฐานที่ใช้ในการเขียนโปรแกรม

1. โครงสร้างแบบลำดับ (Sequence Structure)
 - ❖ ทำงานตามลำดับ
 - ❖ ทำงานจากบนลงล่าง (จุดเริ่มต้นถึงจุดสุด)
 - ❖ มีจุดเริ่มต้นจุดเดียว – จุดสิ้นสุดจุดเดียว
 - ❖ อาจเรียกใช้โมดูลอื่นได้

Flowchart โครงสร้างแบบลำดับ



ตัวอย่าง โปรแกรมรับข้อมูลจำนวนสินค้าและราคасินค้า



ภาพแสดง Flowchart การรับ
จำนวนสินค้าและราคасินค้า

อัลกอริธึม

- เริ่มต้นทำงาน
- กำหนดค่า จำนวนเงินที่ชำระ (Price) เท่ากับศูนย์
- รับข้อมูล จำนวนสินค้า (Amt) และ ราคา (Cost)
- คำนวณจำนวนเงินที่ชำระ จาก $Price = Amt \times Cost$
- แสดงผล จำนวนที่ต้องชำระ
- จบการทำงาน

รหัสเทียม (Pseudo Code)

```

Begin
  Price = 0
  Read Amt, Cost
  Price = Amt x Cost
  Write Price
End
  
```

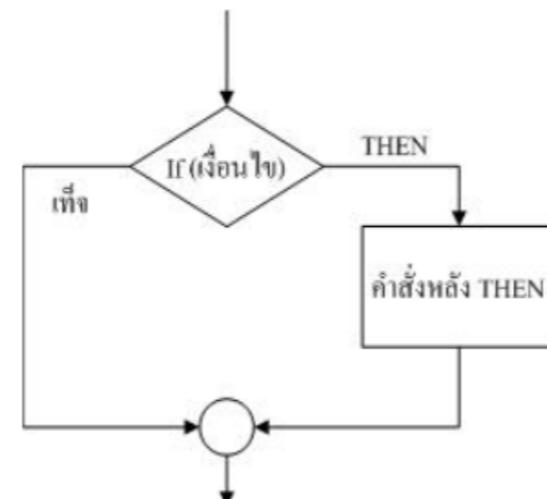
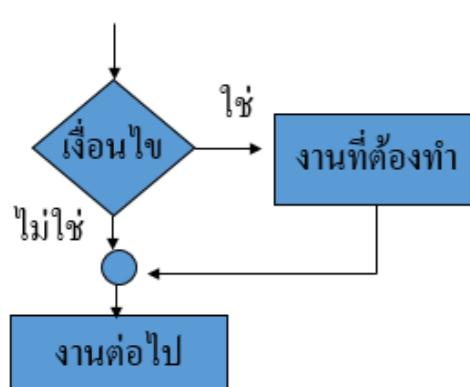
ภาพแสดงการเขียนอัลกอริธึมและรหัสเทียบการรับสินค้าและราคา

2. โครงสร้างแบบเลือก (Selection Structure)

- ❖ มีเงื่อนไขที่ต้องตัดสินใจเลือกการทำงาน
- ❖ ผลลัพธ์ของเงื่อนไขคือ จริง หรือ เท็จ เท่านั้น

2.1 Flowchart แบบหนึ่งทางเลือก หรือ โครงสร้าง IF....THEN

เป็นโครงสร้างที่ทดสอบเงื่อนไข และเลือกว่าจะทำหรือไม่ทำ ก่อนที่จะไปทำงานอื่นต่อไป

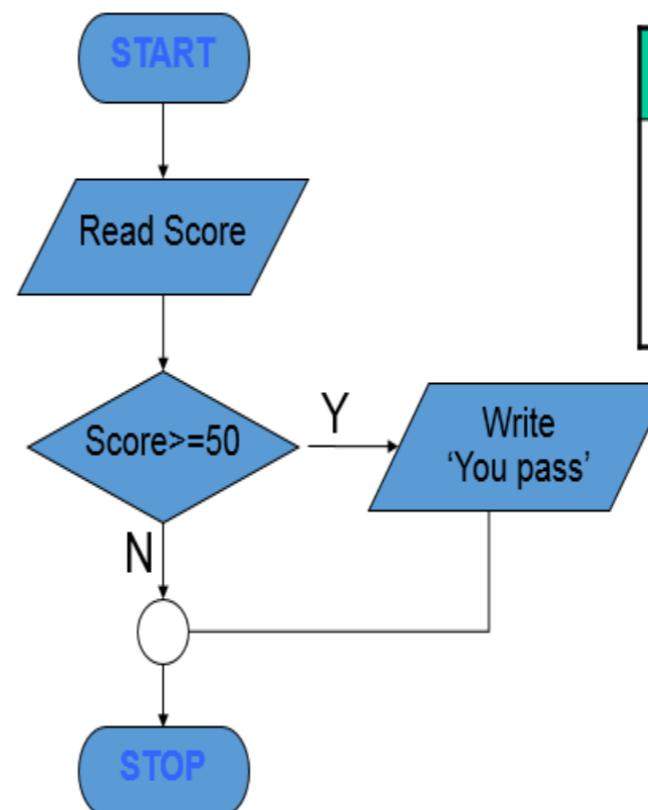


รหัสเทียม Pseudo code

```
IF <เงื่อนไข> THEN  
... (คำสั่งA เมื่อตรวจสอบว่าเงื่อนไขเป็นจริง)...
```

ตัวอย่าง โครงสร้าง IF....THEN

ถ้าได้คะแนนสอบ 50 คะแนนขึ้นไป ให้พิมพ์ข้อความ 'You pass' ถ้าได้คะแนนต่ำกว่า 50 คะแนน ให้จบการทำงาน

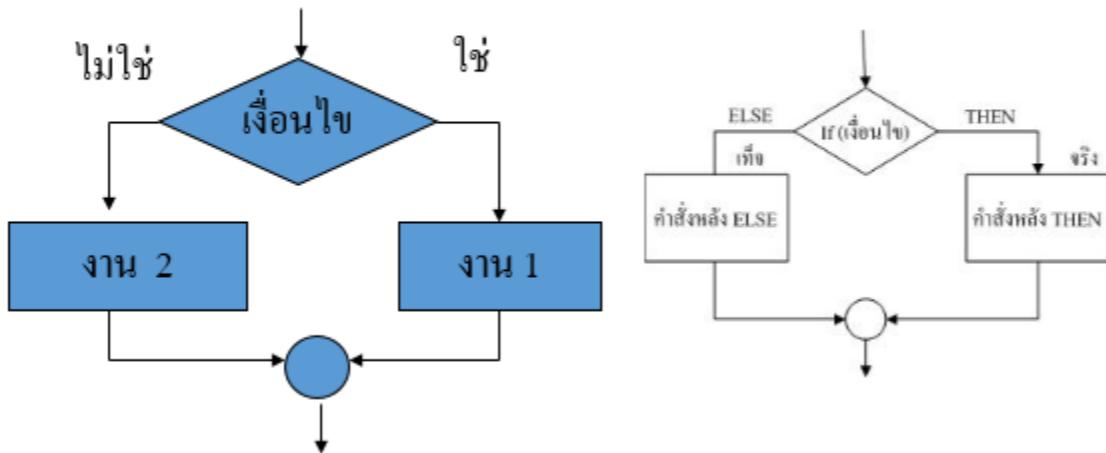


Input	Process	Output
Score	Score > 50	You Pass

ภาพแสดง Flowchart และ อัลกอริธึมของคะแนนสอบ โดยใช้ If

2.2 แบบ 2 ทางเลือก

แบบ 2 ทางเลือก หรือ โครงสร้าง IF...THEN...ELSE



รหัสเทียม Pseudo code

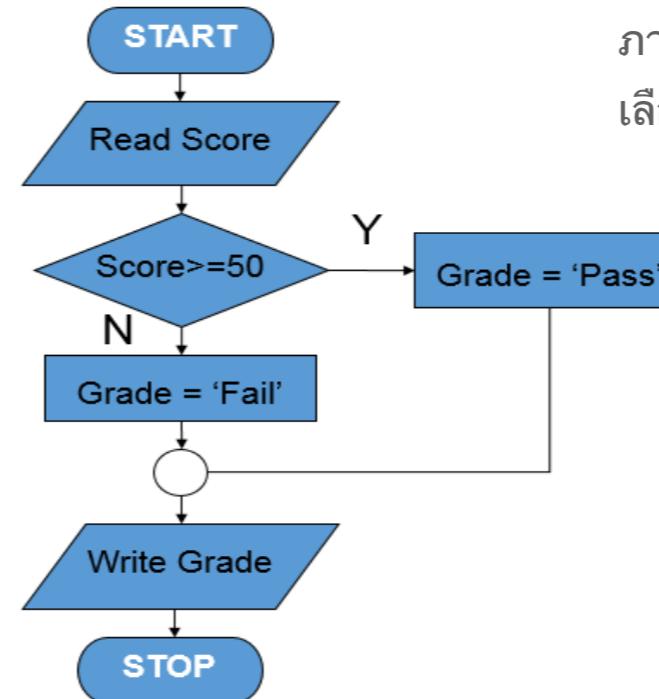
```

IF <เงื่อนไข> THEN
..... (คำสั่งB).....
ELSE ..... (คำสั่งA).....
END IF
    
```

ตัวอย่าง โปรแกรมแสดงผลการสอบทางหน้าจอ

ถ้าได้คะแนนสอบ 50 คะแนนขึ้นไป ให้พิมพ์ข้อความว่า 'Pass'

ถ้าได้คะแนนต่ำกว่า 50 คะแนน ให้พิมพ์ข้อความว่า 'Fail'



ภาพ Flowchart แบบ 2 ทาง
เลือก ของผลการสอบ

- อัลกอริธึม**
- เริ่มต้นทำงาน
 - รับค่า คะแนนสอบ (Score)
 - ถ้าคะแนนสอบตั้งแต่ 50 คะแนน
ขึ้นไปให้กำหนด
 $\text{Grade} = \text{'Pass'}$ ถ้าน้อยกว่า 50
คะแนน ให้กำหนด
 $\text{Grade} = \text{'Fail'}$
 - แสดงผล เกรด (Grade)
 - จบการทำงาน

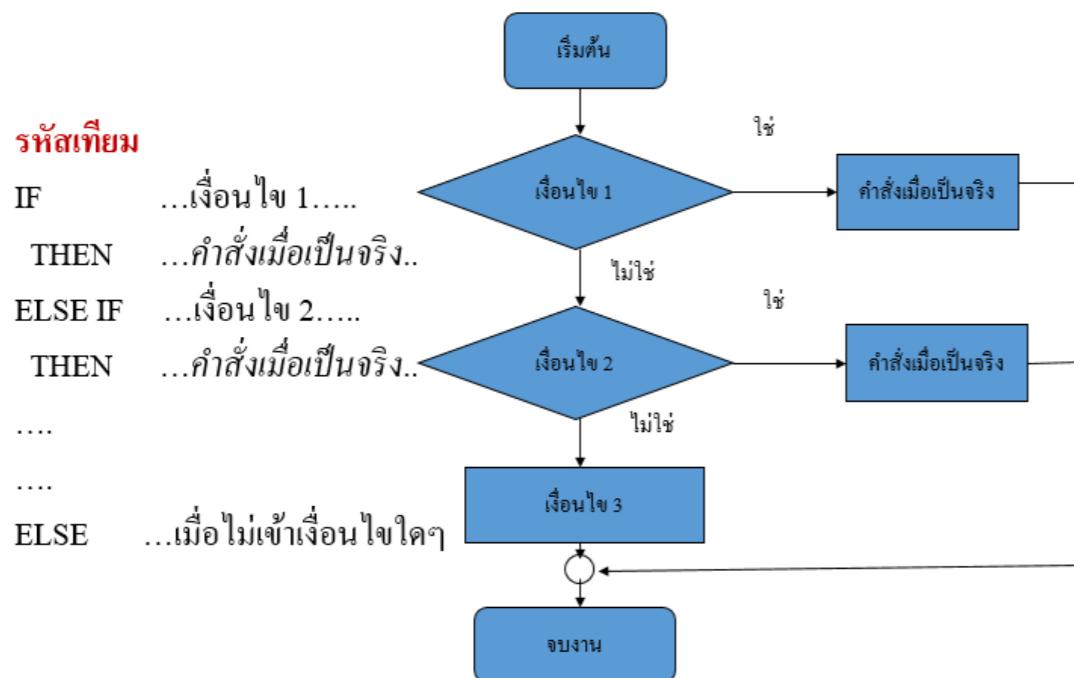
รหัสเทียม (Pseudo Code)

```

Begin
    Read Score
    IF Score >= 50
        THEN Grade = 'Pass'
        ELSE Grade = 'Fail'
    END IF
    Write Grade
End
    
```

ภาพแสดงการเขียนอัลกอริธึมและรหัสเทียบ โดยใช้ if-else

2.3 แบบalogyทางเลือก หรือ โครงสร้าง ELSE...IF



ตัวอย่าง โปรแกรมประมวลผลการเรียน

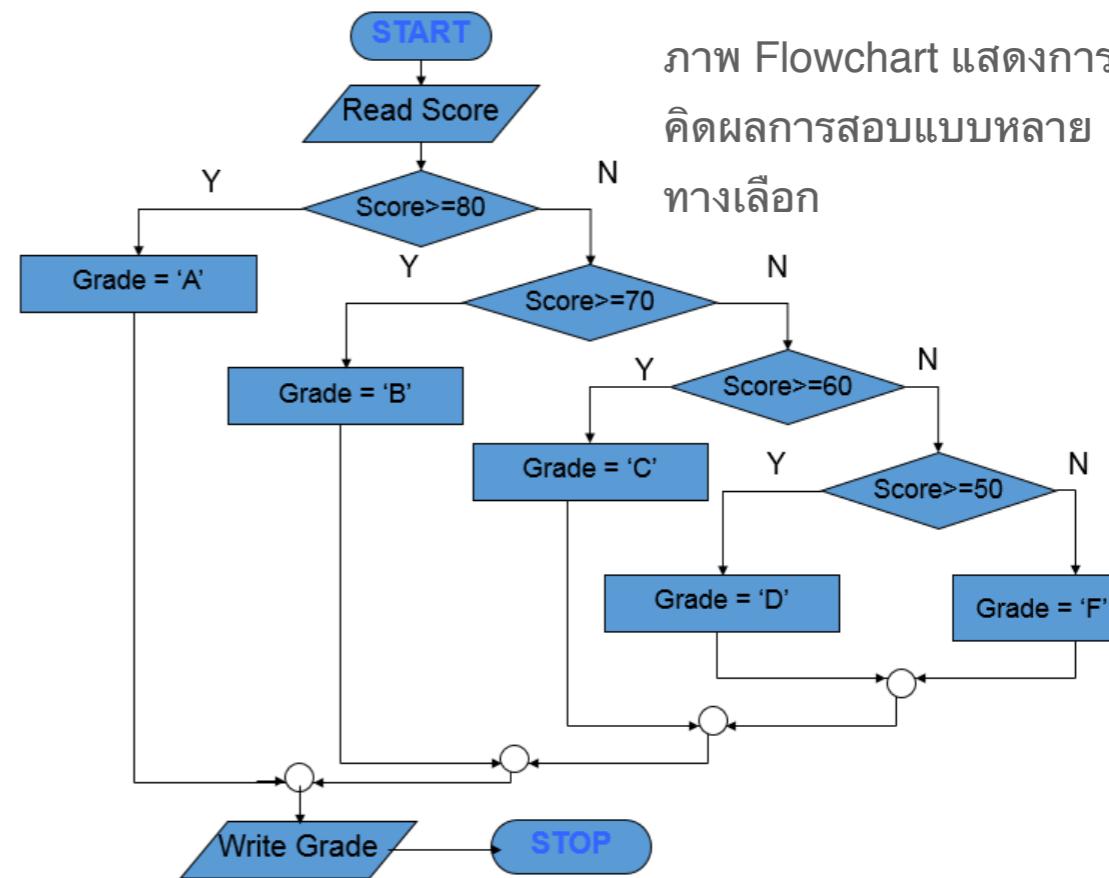
คะแนนสอบสูงกว่า 80 คะแนน ได้เกรด A

คะแนนสอบ 70-79 คะแนนขึ้นไป ได้เกรด B

คะแนนสอบ 60-69 คะแนนขึ้นไป ได้เกรด C

คะแนนสอบ 50-59 คะแนนขึ้นไป ได้เกรด D

คะแนนสอบต่ำกว่า 50 คะแนน ได้เกรด F



อัลกอริธึม

1. เริ่มต้นทำงาน
2. รับค่า คะแนนสอบ (Score)
3. เปรียบเทียบคะแนนสอบ ≥ 80 ถ้าเป็นจริง ให้กำหนด Grade='A' ถ้าเป็นเท็จ ให้เปรียบเทียบคะแนนสอบ ≥ 70 ถ้าเป็นจริง ให้กำหนด Grade='B' ถ้าเป็นเท็จ ให้เปรียบเทียบคะแนนสอบ ≥ 60 ถ้าเป็นจริง ให้กำหนด Grade='C' ถ้าเป็นเท็จ ให้เปรียบเทียบคะแนนสอบ ≥ 50 ถ้าเป็นจริง ให้กำหนด Grade='D' ถ้าเป็นเท็จ ให้กำหนด Grade='F'
4. แสดงผล เกรด (Grade)
5. จบการทำงาน

รหัสเทียม (Pseudo Code)

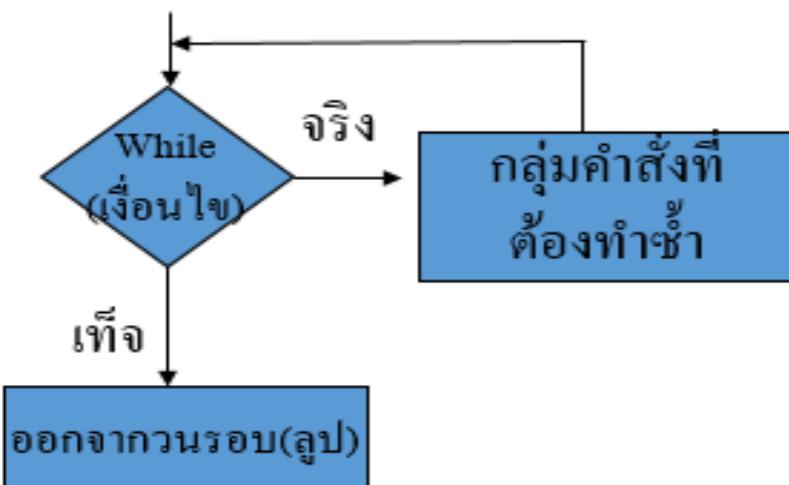
```

Begin
    READ Score
    IF Score >= 80
        THEN Grade = 'A'
    ELSE IF Score >= 70
        THEN Grade = 'B'
    ELSE IF Score >= 60
        THEN Grade = 'C'
    ELSE IF Score >= 50
        THEN Grade = 'D'
    ELSE Grade = 'F'
    WRITE Grade
End
    
```

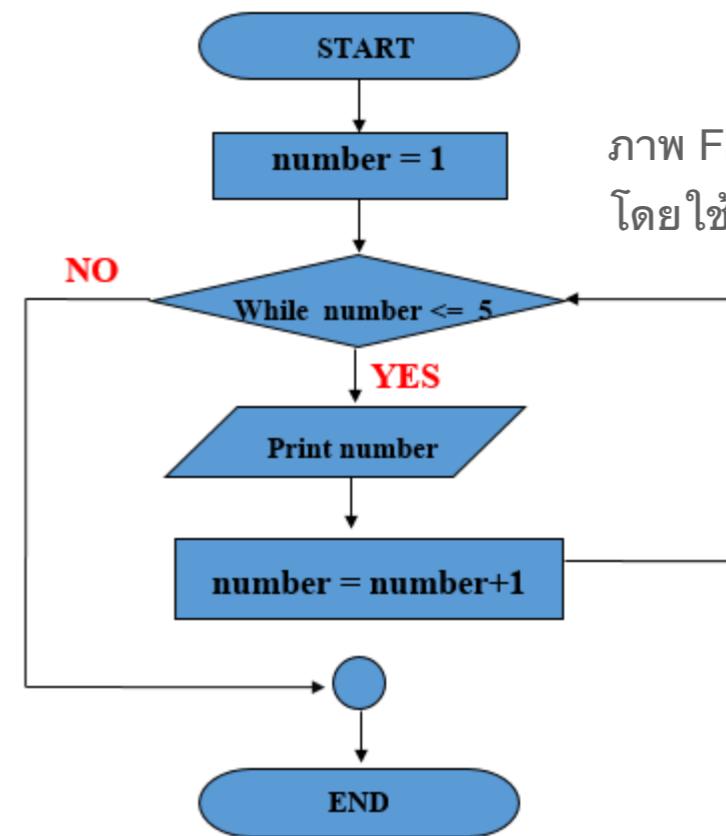
ภาพแสดงอัลกอริธึมและรหัสเทียมของการคิดเกรด ใช้ if-else if

3. โครงสร้างแบบทำซ้ำหรือวนรอบ (Repetition or Looping Structure)

- ❖ การวนซ้ำแบบทดสอบเงื่อนไขก่อน หรือ While
- ❖ มีเงื่อนไขในการหยุดการทำงาน
- ❖ ตรวจเงื่อนไขก่อนการทำงานทุกครั้ง ถ้าใช่ตามที่เงื่อนไขต้องการ จะให้ทำงานต่อไป
- ❖ อาจไม่ได้ทำเลยแม้แต่ครั้งเดียว



ตัวอย่าง โปรแกรมแสดงเลข 1-5



ภาพ Flowchart แสดงเลข 1-5 โดยใช้ While

อัลกอริธึม

- เริ่มต้นทำงาน
- กำหนดค่าเริ่มต้นตัวเลข
Number=1
- ถ้า Number ยังน้อยกว่าหรือเท่ากับ 5 ให้พิมพ์ค่าของ Number และให้เพิ่มค่า Number ครั้งละ 1 ค่า แต่ถ้าเป็นเท็จให้ออกจากการทำงาน
- จบการทำงาน

รหัสเทียม (Pseudo Code)

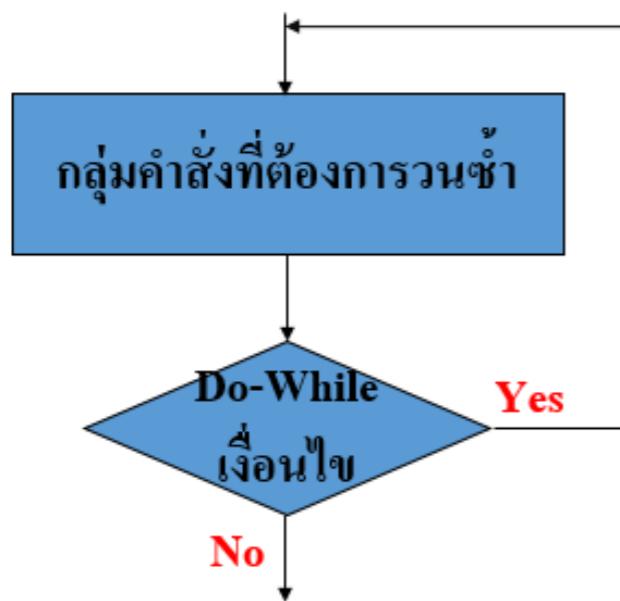
```

Begin
    Number = 1
    While Number <= 5
    {
        Write Number
        Number = Number + 1
    }
End
    
```

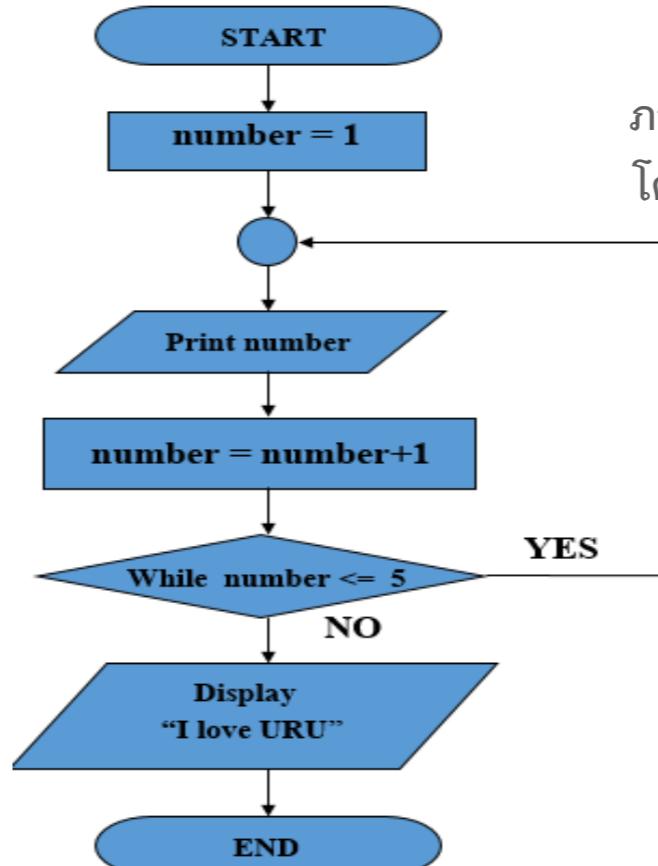
ภาพแสดงอัลกอริธึมและรหัสเทียมการแสดงเลข 1-5 ใช้ While

3.1 การวนซ้ำแบบทดสอบเงื่อนไขทีหลัง หรือ ชนิด Do-While

- ❖ มีเงื่อนไขในการหยุดการทำงาน
- ❖ ได้ทำงานอย่างน้อย 1 ครั้ง
- ❖ หลังจากนั้นในแต่ละครั้งจะมีตรวจสอบเงื่อนไข
- ❖ ถ้าเงื่อนไขเป็นจริงก็จะวนรอบทำงานต่อ ถ้าเป็นเท็จออกจากลูป



ตัวอย่าง แสดงแสดงค่าตัวเลขตั้งแต่เลข 1 ถึงเลข 5



ภาพ Flowchart แสดงเลข 1-5 โดยใช้ Do-While

- อัลกอริธึม
1. เริ่มต้นทำงาน
 2. กำหนดค่าเริ่มต้นตัวเลข
 $number=0$
 3. เพิ่มค่า $number$ ทีละ 1 ค่า
 $number=number+1$ และให้พิมพ์ค่าของ $number$
 4. ถ้าค่าของ $number$ ยังน้อยกว่า หรือเท่ากับ 5 ให้ทำข้อ 3 ไปเรื่อยๆ จนกว่าเงื่อนไขจะเป็นเท็จ และออกจากการทำงาน
 5. จบการทำงาน

รหัสเทียม (Pseudo Code)

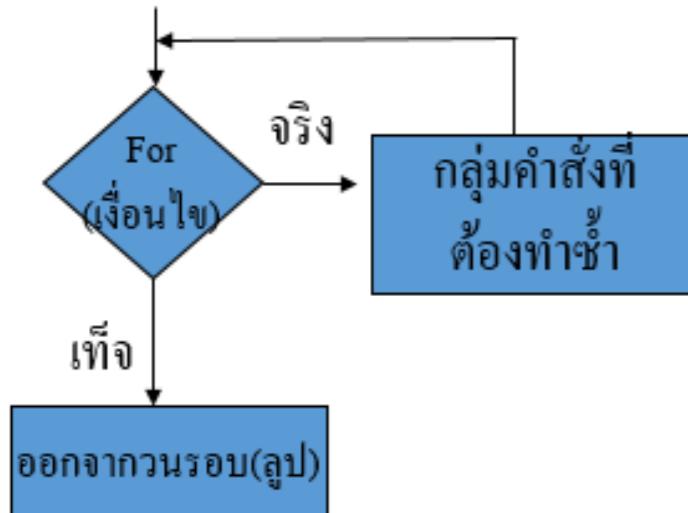
```

Begin
  number = 0
  Do
  {
    number = number+1
    write Number
  }
  While number <=5
End
  
```

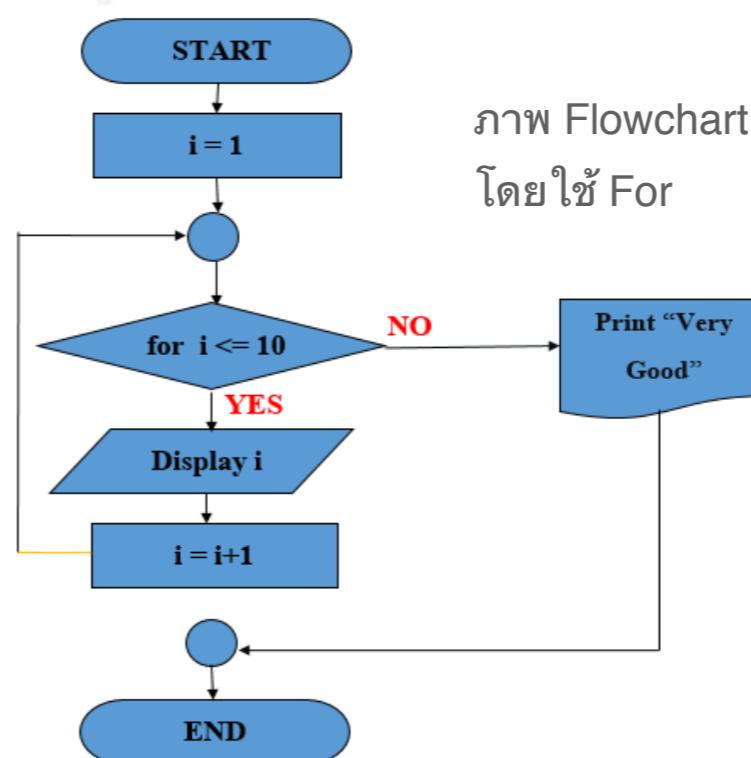
ภาพแสดงอัลกอริธึมและรหัสเทียมแสดงเลข 1-5 ใช้ Do-While

3.2 การวนซ้ำแบบวนซ้ำແນ່ນອນ

- ❖ งานที่ต้องทำซ้ำเป็นจำนวนรอบที่ແນ່ນອນ
- ❖ ไม่มีเงื่อนไขในการหยุดการทำงาน
- ❖ หยุดทำงานเมื่อครบเป็นจำนวนรอบที่ต้องการ
- ❖ มีตัวนับ (Counter) คือคุณคุณจำนวนรอบ



ตัวอย่าง จงเขียนผังงานแสดงการทำงานของการแสดงค่าตัวเลขตั้งแต่เลข 1 ถึงเลข 10



ภาพ Flowchart แสดงเลข 1-10 โดยใช้ For

อัลกอริธึม

1. เริ่มต้นทำงาน
2. กำหนดค่าเริ่มต้นตัวเลข $i=1$
3. ถ้า i ยังน้อยกว่าหรือเท่ากับ 10
ให้พิมพ์ค่า i และเพิ่มค่า i ทีละ 1
ค่า $i=i+1$
4. ทำข้อ 3 ไปเรื่อยๆ จนกว่าค่าของ i จะเป็นเท็จ
5. พิมพ์ข้อความ “Very Good”
6. จบการทำงาน

รหัสเทียบ (Pseudo Code)

```

Begin
    i=1
    for i<=10
    {
        Write i
        i=i+1
    }
    Write "Very Good"
End
    
```

ภาพแสดงอัลกอริธึมและรหัสเทียบแสดงเลข 1-10 โดยใช้ For

แบบฝึกหัด

1. จงเขียน Algorithm, Pseudo code, Flowchart ของโปรแกรมที่กำหนดให้ต่อไปนี้

1.1 โปรแกรมคำนวณหาค่า y ของสมการ $y = x^2 + 2x + 10$

1.2 โปรแกรมแสดงยอดขาย ถ้าชื่อสินค้ามากกว่า 1000 บาท มีส่วนลดให้ 100 บาท

1.3 โปรแกรมแสดงยอดขายของร้านค้าแห่งหนึ่งมีราย ลดราคาให้ลูกค้า ถ้าเป็นชายจะลดให้ 50 บาท แต่ถ้าเป็นหญิง จะลดให้ 100 บาท

1.4 โปรแกรมแสดงขนาดของการใช้ยาตามอายุของผู้ใช้

อายุมากกว่า 10 ปี แสดงข้อความรับประทานครั้งละ 3 ช้อนชา

อายุ 6-10 ปี แสดงข้อความรับประทานครั้งละ 2 ช้อนชา

อายุ 2-5 ปี แสดงข้อความรับประทานครั้งละ 1 ช้อนชา

เด็กอายุต่ำกว่า 1 ปี ห้ามรับประทาน

2. จงเขียน Flowchart สำหรับรับค่าตัวเลข และแสดงค่าตัวเลขที่รับเข้ามาอุ่นทางจอกาพ โดยที่เมื่อรับค่าตัวเลขเป็น -999 จะลิ้นสูดการวนรอบและจบโปรแกรม

3. จงเขียน Flowchart สำหรับรับค่าตัวเลข และในระหว่างรับให้หาผลรวมของตัวเลขที่รับเข้ามา โปรแกรมจะหยุดรับค่าเมื่อใส่ตัวเลข -999 และจะแสดงผลรวมของตัวเลขทั้งหมดที่รับเข้ามาและหยุดทำงาน

4. จงเขียน Algorithm แบบทำงานซ้ำมา 2 ตัวอย่าง

ตัวอย่างการเขียนการบ้าน

จงเขียนอัลกอริธึมเพื่อรับข้อมูลจำนวนสินค้า และราคาสินค้า เพื่อคำนวณและแสดงผลจำนวนเงินที่ต้องชำระ

Input	Process	Output
Read: Amt , Cost	Price = Amt * Cost	Print:Price

1. Output: จำนวนเงินที่ต้องชำระ (Price)

2. Input: จำนวนสินค้า (Amt) ราคาสินค้า (Cost)

3. Process

3.1 เริ่มต้นการทำงาน

3.2 กำหนดค่าเริ่มต้นให้ Price = 0

3.3 รับข้อมูลจำนวนสินค้า (Amt) และราคาสินค้า (Cost)

3.4 คำนวณจำนวนเงินที่ต้องชำระ Price = Cost * Amt

3.5 แสดงผลจำนวนเงินที่ต้องชำระ (Price)

3.6 จบการทำงาน

อัลกอริทึม	ชุดโค้ด
3.1 เริ่มต้นการทำงาน	Begin
3.2 กำหนดค่าเริ่มต้นให้ราคาสินค้า (Price) =0	Price = 0
3.3 รับข้อมูลจำนวนสินค้า(Amt) และราคาสินค้า(Cost)	Read Amt , Cost
3.4 คำนวณเงินที่ต้องชำระ Price = Amt * Cost	Price = Amt * Cost
3.5 แสดงจำนวนเงินที่ต้องชำระ	Write Price
3.6 จบการทำงาน	End



BIG

วัตถุประสงค์

- 1.เข้าใจการวัดประสิทธิภาพของอัลกอริธึม
- 2.เข้าใจสภาวะของอัลกอริธึม

Section 1

ประสิทธิภาพของอัลกอริธึม

ในการเขียนโปรแกรมคอมพิวเตอร์สิ่งที่สำคัญที่สุดที่จะต้องคำนึงถึง คือ ผลลัพธ์ของการประมวลผล โดยจะต้องมีความถูกต้อง แม่นยำ สามารถแก้ปัญหาได้ตรงตามที่ตั้งใจไว้

ลักษณะสำคัญของ โปรแกรมคอมพิวเตอร์ที่ดีไม่ควรใช้เวลาในการประมวลผลนานจนเกินไป ซึ่งการที่จะทำให้โปรแกรมคอมพิวเตอร์ลดเวลาในการประมวลผลลงได้นั้น จะต้องออกแบบอัลกอริธึม ให้มีประสิทธิภาพ

ประสิทธิภาพของอัลกอริธึม จะพิจารณาอยู่ 2 ส่วนหลัก ๆ ได้แก่

- ❖ หน่วยความจำ (Memory) ที่จะต้องใช้ในการประมวลผล
- ❖ เวลา (Time) ที่จะต้องใช้ในการประมวลผล

โดยทั้ง 2 สิ่งนี้มักถูกใช้เป็นตัวตัดสินประสิทธิภาพของอัลกอริธึมว่า มีประสิทธิภาพมากหรือน้อยเพียงใด ดังนั้น ถ้าจำเป็นจะต้องดำเนินการแก้ไขปรับปรุงเพื่อเพิ่มประสิทธิภาพของอัลกอริธึม จำเป็นจะต้องพิจารณา ก่อนว่าจะสามารถ

ดำเนินการแก้ไขกับส่วนใดได้บ้าง และส่งผลกระทบต่อกันหรือไม่ อย่างไร

การประเมินประสิทธิภาพของอัลกอริธึม

การประเมินประสิทธิภาพของอัลกอริธึมแบ่งออกเป็น 2 วิธี คือ การวิเคราะห์และวัดผล ดังนี้

1. การวิเคราะห์ประสิทธิภาพของอัลกอริธึม (Performance Analysis) จะใช้วิธีการวิเคราะห์วิธีการทำงานของอัลกอริธึม
2. การวัดประสิทธิภาพของอัลกอริธึม (Performance Measurement) เป็นการวัดผลจากการทดลองจริง

การวิเคราะห์ประสิทธิภาพของอัลกอริธึม

การวิเคราะห์ประสิทธิภาพของอัลกอริธึมแบ่งออกเป็น 2 ส่วน ดังนี้

1. การวิเคราะห์หน่วยความจำที่ต้องใช้ในการประมวลผล (Space Complexity)

2. การวิเคราะห์เวลาที่ต้องใช้ในการประมวลผล (Time Complexity)

การวิเคราะห์ Space Complexity

การวิเคราะห์ Space Complexity ของอัลกอริทึมคือการวิเคราะห์ว่าจะต้องใช้หน่วยความจำทั้งหมดเท่าไรในการประมวลผลอัลกอริทึมนั้น สาเหตุที่ต้องทราบจำนวนของหน่วยความจำที่จะต้องใช้นั้นมีเหตุผลดังนี้

- ❖ ทำให้ทราบว่าอัลกอริทึมนั้นสามารถรองรับจำนวนข้อมูลที่ส่งเข้ามาประมวลผล (Input Data) ได้มากที่สุดเท่าใด เพื่อให้อัลกอริทึมนั้นสามารถประมวลผลได้อยู่
- ❖ กรณีที่ต้องประมวลผลบนเครื่องคอมพิวเตอร์ที่ใช้งานร่วมกันหลายคนในเครือข่าย จะเป็นจะต้องทราบขนาดของหน่วยความจำที่จะต้องใช้ในการประมวลผลอัลกอริธึม เพื่อไม่ให้กระทบกับการทำงานของคนอื่น
- ❖ เพื่อเลือกคุณลักษณะของคอมพิวเตอร์ที่จะใช้ติดตั้งโปรแกรมที่พัฒนาขึ้นได้อย่างเหมาะสม เพื่อถ้านำไปติดตั้งที่เครื่องที่มีหน่วยความจำไม่เพียงพอ โปรแกรมก็จะไม่ทำงาน

องค์ประกอบของ Space Complexity

การวิเคราะห์ Space Complexity ประกอบด้วย 3 ส่วนคือ Instruction Space, Data Space และ Environment Stack Space ดังนี้

1) Instruction Space

คือ จำนวนของหน่วยความจำที่คอมไพล์เร็วจำเป็นต้องใช้ขณะทำการคอมไพล์โปรแกรม ซึ่งจำนวนหน่วยความจำที่ต้องใช้จะขึ้นอยู่กับคอมไพล์เร็วแต่ละประเภท

2) Data Space

คือ จำนวนหน่วยความจำที่ต้องใช้สำหรับเก็บค่าคงที่ และตัวแปรทั้งหมดที่ต้องใช้ในการประมวลผล โปรแกรม ซึ่ง Data Space แบ่งออกเป็น 2 ประเภท คือ

❖ หน่วยความจำแบบ static คือ จำนวนของหน่วยความจำที่ต้องใช้อย่างแน่นอน ไม่มีการเปลี่ยนแปลง ประกอบด้วยหน่วยความจำที่ใช้เก็บค่าคงที่และตัวแปรประเภท array

❖ หน่วยความจำแบบ dynamic คือ จำนวนของหน่วยความจำที่ใช้ในการประมวลผลสามารถเปลี่ยนแปลงได้ และจะทราบจำนวนหน่วยความจำที่จะใช้ก็ต่อเมื่อโปรแกรมกำลังทำงานอย

3) Environment Stack Space

คือ หน่วยความจำที่ต้องใช้ในการเก็บผลลัพธ์ของข้อมูล เอาไว้ เพื่อรอเวลาที่จะนำผลลัพธ์นั้นกลับไปประมวลผลอีกครั้ง หน่วยความจำประเภทนี้จะเกิดขึ้นเมื่อมีการร้องขอให้นำมาใช้ เท่านั้น

4) เทคนิคการเขียนโปรแกรมคอมพิวเตอร์ที่ต้องร้องขอ พื้นที่ในหน่วยความจำประเภทนี้มาใช้ก็คือ การทำ Recursive โดยหน่วยความจำที่ต้องใช้ก็จะขึ้นอยู่กับความลึกของการทำ Recursive ด้วย ยิ่งลึกมากก็จะยิ่งใช้หน่วยความจำมากขึ้น ตามไปด้วย

ตัวอย่างการวิเคราะห์ Space Complexity

```
{  
    int num1, num2, temp;  
    temp = num1;  
    num1 = num2;  
    num2 = temp;  
}
```

จากตัวอย่าง ตัวแปรที่ใช้จะมีอยู่ด้วยกัน 3 ตัวแปร ประกอบด้วย num1, num2 และ temp ซึ่งเป็นตัวแปรชนิด integer ทั้งหมด โดยจะใช้หน่วยความจำในการเก็บตัวแปรประเภท integer ตัวแปรละ 2 bytes ดังนั้น จะต้องใช้หน่วยความจำทั้งสิ้น $3 \times 2 = 6$ bytes

```
int factorial(int n)  
{  
    if (n==0)  
        return 1;  
    else  
        return (n * factorial(n-1));  
}
```

ในตัวอย่างนี้มีตัวแปรที่ใช้ 1 ตัวคือ n ซึ่งเป็นตัวแปรชนิด integer แต่ลักษณะการทำงานเป็นแบบ recursive ดังนั้น หน่วยความจำส่วนใหญ่จะไปขึ้นอยู่กับความลึกของการทำ recursive ซึ่งจะเห็นว่า การประมวลผลของคำสั่ง if จะมีอยู่ด้วย กัน 2 แบบ คือ เมื่อ n = 0 และเมื่อ n มีค่าใดๆ ดังนั้น ความลึกของการทำ recursive จะมีค่าเท่ากับ 1 (กรณีเมื่อ n = 0) หรือมีค่าเท่ากับ n (เมื่อ n เป็นค่าใดๆ) เท่านั้น

สรุปออกมาได้ว่า **ความลึกของการทำ Recursive คือ ค่าที่มากที่สุดระหว่าง 1 กับ n สามารถแทนได้ด้วยสัญลักษณ์ Max {1, n}**

ขั้นตอนไปจะพิจารณาว่า แต่ละครั้งที่มีการเรียกใช้ฟังก์ชัน Factorial จะต้องใช้หน่วยความจำในการเก็บข้อมูลเท่าไร ซึ่ง จากตัวอย่างจะต้องใช้หน่วยความจำทั้งสิ้น 4 bytes (สำหรับเก็บ address 2 bytes และตัวแปรชนิด integer อีก 2 bytes) ทำให้สามารถสรุปได้ว่า ต้องใช้หน่วยความจำสำหรับอัลกอริธึมที่ 2 ทั้งหมดเท่ากับ $4 \times \text{Max}\{1, n\}$ bytes

การวิเคราะห์ Time Complexity

Time Complexity คือ เวลาที่เครื่องคอมพิวเตอร์ต้องใช้ในการประมวลผลอัลกอริธึม เหตุผลที่ควรทราบเวลาที่ต้องใช้มีหลายประการด้วยกัน ยกตัวอย่างเช่น

- ❖ ทำให้สามารถประมาณเวลาทั้งหมดที่ต้องใช้ในโปรแกรม
 - ❖ สามารถมุ่งประเด็นการแก้ไขไปที่อัลกอริธึมที่ใช้เวลาในการประมวลผลนานๆ ทำให้ไม่ต้องแก้ไขทั้งโปรแกรม
 - ❖ โปรแกรมคอมพิวเตอร์ที่ทำงานแบบ Interactive เวลาที่ใช้ในการประมวลผลแต่ละขั้นตอนจะเป็นสิ่งสำคัญ เพราะผู้ใช้ไม่ควรรอการประมวลผลเป็นเวลานานในการ Interactive แต่ละครั้ง
 - ❖ สามารถเลือกคุณลักษณะของคอมพิวเตอร์ที่จะใช้ติดตั้งโปรแกรมที่พัฒนาขึ้นได้อย่างเหมาะสม
- หลักในการพิจารณาอย่างคร่าวๆ ถึงเวลาที่ต้องใช้ในการประมวลผล เช่น
- ❖ โปรแกรมจะสามารถประมวลผลได้เร็วกว่า เมื่ออุปกรณ์เครื่องคอมพิวเตอร์ที่มีความเร็วในการประมวลผลสูงกว่า

❖ ถ้าใช้คอมไพเลอร์ตัวเดียวกัน code ที่สั้นกว่า ย่อมใช้เวลาในการประมวลผลได้น้อยกว่า

เวลาในการประมวลผลของโปรแกรม

❖ Compile Time คือ เวลาที่ใช้ในการตรวจสอบไวยากรณ์ (syntax) ของ code ว่าเขียนได้ถูกต้องหรือไม่ ถ้ามีข้อผิดพลาดเกิดขึ้นคอมไพเลอร์จะแจ้งเตือนให้ทราบ

❖ Run Time หรือ Execution Time คือ เวลาที่เครื่องคอมพิวเตอร์ใช้ในการประมวลผล

การนับตัวดำเนินการ

ในการวิเคราะห์ Time Complexity วิธีที่มักใช้กันบ่อยๆ คือ การนับตัวดำเนินการ (Operation count) ในอัลกอริธึม โดยพิจารณาลักษณะของตัวดำเนินการควบคู่ไปด้วย สามารถสรุปอุปกรณ์ได้เป็นหลักคร่าวๆ ดังนี้

1) แบบ Linear Loops

อัลกอริทึมมีการทำงานแบบวนรอบ (Loop) โดยแต่ละ loop จะมีการเพิ่มหรือลดค่า ในปริมาณที่คงที่ เช่นเพิ่มค่าตัวแปรขึ้นทีละหนึ่ง ในแต่ละรอบ เช่น

```
x = 1  
Loop (x <= 2000)  
    x = x+1
```

```
x = 1  
Loop (x <= 2000)  
    x = x+5
```

ถ้าให้ $f(n)$ แทนประสิทธิภาพ และ n แทนจำนวนรอบการทำงาน สามารถเขียนเป็นสมการวัดประสิทธิภาพของอัลกอริธึมแบบ Linear loop ได้ดังนี้

$$f(n) = n$$

2) แบบ Logarithmic Loops

อัลกอริธึมจะทำงานแบบ Loop โดยการทำงานภายในแต่ละ loop จะเพิ่มหรือลดค่าเป็นเท่าตัว เช่น คูณเพิ่มค่าตัวแปรขึ้นทีละ 2 เท่า ในแต่ละรอบ

```
x = 1  
Loop (x < 1000)  
    x = x*2
```

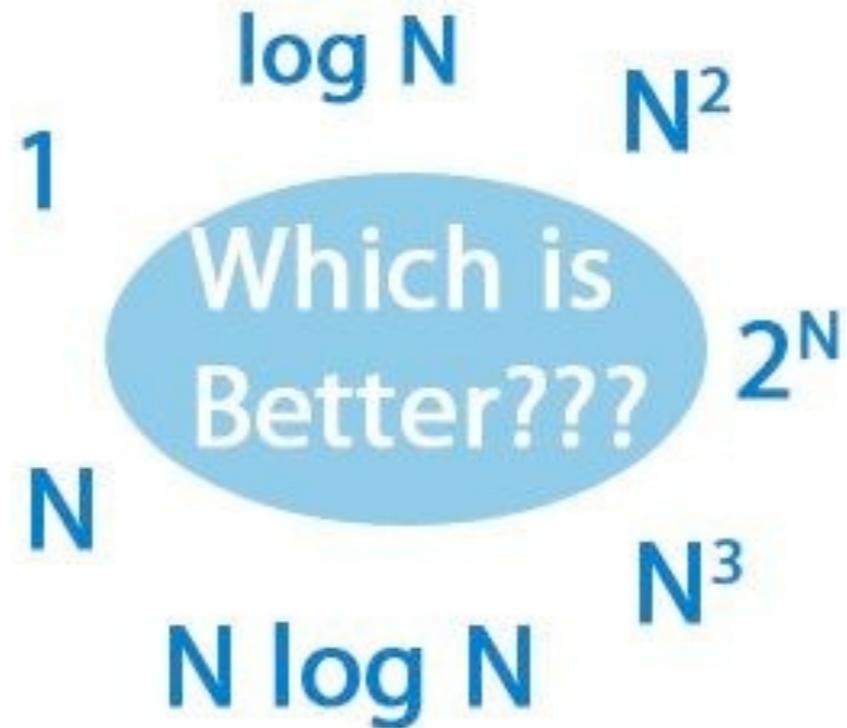
```
x = 1000  
Loop (x >= 1)  
    x = x/2
```

ถ้าให้ $f(n)$ แทนประสิทธิภาพ และ n แทนจำนวนรอบการทำงาน สามารถเขียนเป็นสมการวัดประสิทธิภาพของอัลกอริธึมแบบ Linear loop ได้ดังนี้

$$f(n) = [log n]$$

3) แบบ Nested Loops

คือ ลักษณะของอัลกอริธึมที่มี loop ช้อนอยู่ภายใน loop โดยประสิทธิภาพของอัลกอริธึมก็จะมีค่าเท่ากับจำนวน loop ทั้งหมดที่จะต้องประมวลผล ซึ่งหาได้จากการเอาจำนวน loop ที่ช้อนกันมาคูณกัน



สภาพประสิทธิภาพของอัลกอริธึม

ลักษณะของกลุ่มข้อมูลที่เข้ามาประมวลผล สามารถที่จะแบ่งประสิทธิภาพของอัลกอริธึมออกเป็น 3 สภาวะด้วยกัน ซึ่งในการวิเคราะห์ประสิทธิภาพของอัลกอริธึมเราจะต้องมีการระบุด้วยว่า ค่าผลลัพธ์ที่เราได้จากการวิเคราะห์นั้น พิจารณาเมื่ออัลกอริธึมอยู่ในสภาวะใด ซึ่งแบ่งออกเป็น

❖Best-case

คือ การวิเคราะห์อัลกอริธึมเมื่อข้อมูลที่เข้ามาประมวลผล ส่งผลให้อัลกอริธึมนี้มีประสิทธิภาพดีที่สุด และใช้เวลาในการประมวลผลน้อยที่สุดด้วย

❖Worst-case

คือ การวิเคราะห์อัลกอริธึมเมื่อข้อมูลที่เข้ามาประมวลผล ส่งผลให้อัลกอริธึมนี้มีประสิทธิภาพแย่ที่สุด และใช้เวลาในการ

ประมาณการที่สุดด้วย ซึ่งกรณีค่าที่ได้จากการวิเคราะห์ ต้องมีค่ามากที่สุดเมื่อเทียบกับกรณีอื่น

❖Average-case

คือ การวิเคราะห์อัลกอริทึมเมื่อข้อมูลที่เข้ามาประมาณผลลัพธ์ให้อัลกอริทึมโดยเฉลี่ยมีค่าประมาณที่สามารถระบุได้

โดยส่วนใหญ่มักนิยมเปรียบเทียบประสิทธิภาพของอัลกอริทึมในสภาวะ Worst-case เนื่องจากเป็นสภาวะที่ใช้หน่วยความจำมากที่สุดและใช้เวลาในการประมาณการที่สุด จึงถือเป็นอีกสิ่งหนึ่งที่ใช้รับประกันได้ว่า อัลกอริทึมนั้นๆ จะไม่ทำงาน慢ๆ ไปกว่าค่านี้อีกแล้ว ใช้ในการรับประกันประสิทธิภาพของอัลกอริทึม

สัญลักษณ์ Asymptotic

สัญลักษณ์ Asymptotic คือ พังก์ชันของเวลาที่ใช้ในการประมาณอัลกอริทึมนั้น ๆ โดยพิจารณาค่าเมื่ออัลกอริทึมนั้นมีปริมาณข้อมูลมาก ๆ สมมติให้มีค่าเป็นอนันต์ (Infinity) แล้วพิจารณาว่า ต้องใช้เวลาในการประมาณมีขอบเขตของแนวโน้มการเติบโตทางเวลา (Growth in Run Time) เป็นอย่างไร ซึ่งเขียนแทนออกมายเป็นพังก์ชันทางคณิตศาสตร์ หรืออาจกล่าวได้ว่า เพื่อเป็นการอธิบายให้เห็นภาพรวมของพฤติกรรมทางเวลาของอัลกอริทึมนั้น ๆ

พังก์ชันการเติบโตทางเวลา (Growth in Run Time)

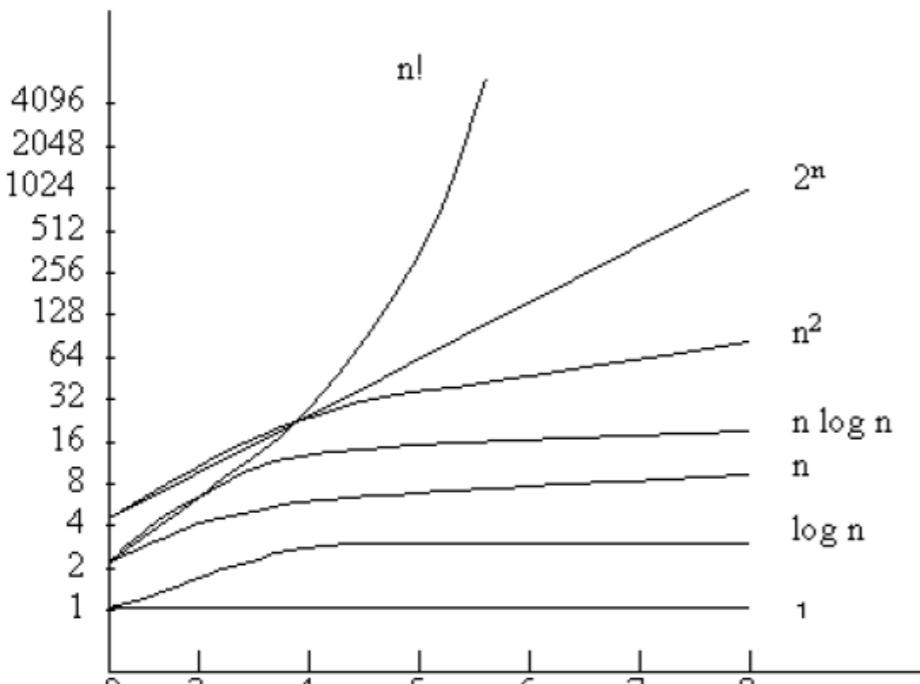
ค่าพังก์ชันที่ใช้อธิบายถึงพฤติกรรมแนวโน้มการเติบโตทางเวลาของอัลกอริทึม หรือพังก์ชันเติบโตทางเวลา (Growth in Run Time) นั้น เป็นค่าพังก์ชันทางคณิตศาสตร์ที่แสดงถึงความสัมพันธ์ระหว่างปริมาณข้อมูลที่กำลังประมาณผล กับเวลาที่ต้องใช้ในการประมาณข้อมูลนั้น ให้เสร็จว่ามีความสัมพันธ์กันอย่างไร โดยการคำนวณหาค่า Run Time ของอัลกอริทึมนั้น ๆ เราสามารถหาได้จากการคำนวณ Step Counts

การคำนวณ Step Counts จะพิจารณาทางงานทั้งหมดที่อัลกอริทึมนั้นต้องทำ นั่นคือ หาว่าจะต้องประมาณผลคำสั่งทั้งหมดที่อยู่ภายใต้อัลกอริทึมเป็นจำนวนรวมทั้งสิ้นกี่ครั้ง ตัวอย่างการคำนวณค่า Step Counts

code	จำนวนครั้ง(step)ที่ประมาณผล
void main()	0
{	1
int i, n;	1
scanf("%d",&n);	
for(i=1;i<n;i++)	1+n+n
{	
printf("hello");	n
printf("\n");	n
}	
}	
รวมทั้งสิ้น (ค่า step counts)	4n+3

สัญลักษณ์ Big-Oh หรือ O ()

สัญลักษณ์ Big-Oh เป็นการอธิบายถึงขอบเขตบนของฟังก์ชันการเติบโตทางเวลา (Upper-bound function) สำหรับอัลกอริธึมหนึ่ง ๆ ว่าเวลาที่ใช้ในการประมวลผลสูงสุดไม่ว่า N จะมีค่าเป็นเท่าไรก็ตาม จะใช้เวลาไม่เกินเวลาที่คำนวณได้จากฟังก์ชันนี้ สัญลักษณ์ที่ใช้สื่อความหมายคือ O ())



แสดงการเติบโตของฟังก์ชันที่ใช้บ่อยๆในการคำประมาณ Big-O

	$\log n$	n	$n \log n$	n^2	n^3	2^n
1	3 nsec	0.01 μ	0.02 μ	0.06 μ	0.51 μ	0.26
2	4 nsec	0.02 μ	0.06 μ	0.26 μ	4.10 μ	65.5
3	5 nsec	0.03 μ	0.16 μ	1.02 μ	32.7 μ	4.29 s
4	6 nsec	0.06 μ	0.38 μ	4.10 μ	262 μ	5.85 c
5	0.01 μ	0.13 μ	0.90 μ	16.38 μ	0.01 sec	10^{20} c
6	0.01 μ	0.26 μ	2.05 μ	65.54 μ	0.02 sec	10^{58} c
7	0.01 μ	0.51 μ	4.61 μ	262.14 μ	0.13 sec	10^{135} c
8	0.01 μ	2.05 μ	22.53 μ	0.01 sec	1.07 sec	10^{598} c
9	0.01 μ	4.10 μ	49.15 μ	0.02 sec	8.40 sec	10^{1214} c
10	0.01 μ	8.19 μ	106.50 μ	0.07 sec	1.15 min	10^{2447} c
11	0.01 μ	16.38 μ	229.38 μ	0.27 sec	1.22 hrs	10^{4913} c
12	0.02 μ	32.77 μ	491.52 μ	1.07 sec	9.77 hrs	10^{9845} c
13	0.02 μ	65.54 μ	1048.6 μ	0.07 min	3.3 days	10^{19709} c
14	0.02 μ	131.07 μ	2228.2 μ	0.29 min	26 days	10^{39438} c
15	0.02 μ	262.14 μ	4718.6 μ	1.15 min	7 mnths	10^{78894} c
16	0.02 μ	524.29 μ	9961.5 μ	4.58 min	4.6 years	10^{157808} c
17	0.02 μ	1048.60 μ	20972 μ	18.3 min	37 years	10^{315634} c

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do tempor incididunt ut labore et dolore magna aliqua.

• •

Section 2

แบบฝึกหัด

1. จงเรียงลำดับฟังก์ชันการทำงานต่อไปนี้ตามอัตราการเจริญเติบโต

0.5^n , 1 , $\log n$, n , 10^n

2. จงเปรียบเทียบอัตราการเจริญเติบโตของ
 n^{10} , 2^n

3. จงหาค่า BigO ของ

n^3+2n^3+10

4. จงหาค่า BigO ของ 100

5. จงหาค่า BigO ของ $100N + 1$

6. จงหาค่า BigO ของ $20n\log n + 5N$

7. สมมติให้แต่ละโปรแกรมใช้เวลาในการทำงานดังนี้

$$\text{prg1} = 3n^2 + 2n$$

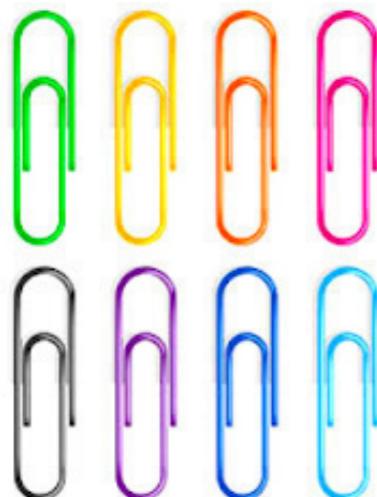
$$\text{prg2} = 2\log_2 n + 6n + n$$

$$\text{prg3} = n + n\log_2 n + 4n + 9$$

จงแสดง BigO ของแต่ละโปรแกรมพร้อมทั้งเรียงลำดับประสิทธิภาพของโปรแกรมจากดีสุดไปช้าสุด

Array Structure

GUESS MY ARRAY!



2 rows of 4

$$2 \times 4 = 8$$



3 rows of 3

$$3 \times 3 = 9$$



4 rows of 8

$$4 \times 8 = 32$$

วัตถุประสงค์

1. บอกรู้สัมบัติ โครงสร้างข้อมูลแบบอาร์เรย์ได้
2. สามารถอ้างอิงตำแหน่งสมาชิกและคำนวณหาสมาชิกในอาร์เรย์ได้
3. เข้าใจหลักการเก็บอาร์เรย์ในหน่วยความจำ
4. สามารถคำนวณหาผลเดรลส์ในหน่วยความจำของอาร์เรย์ได้

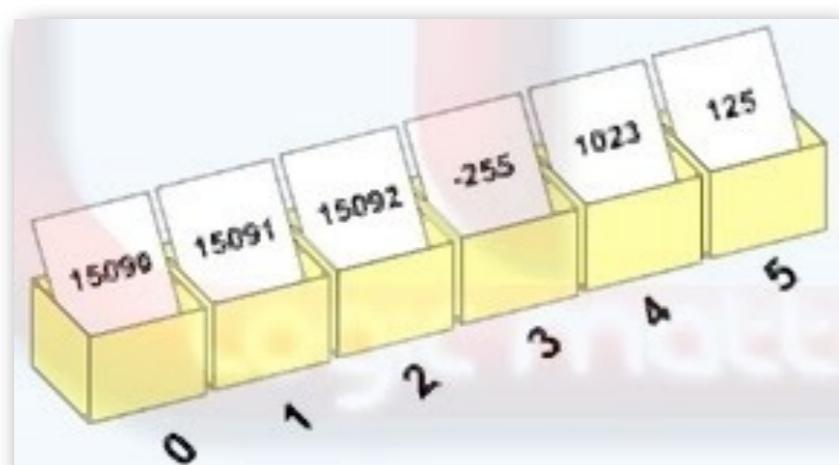
นิยามโครงสร้างอาร์เรย์ (Array)

อาร์เรย์ (Array) เป็นโครงสร้างข้อมูลที่ใช้เก็บข้อมูลชนิดเดียวกัน เป็นกลุ่มหรือชุดที่เรียงติดต่อ กันเป็น列า มีขอบเขตจำกัดและมีขนาดคงที่

ข้อมูลชนิดเดียวกัน คือ ข้อมูลทุกตัวที่อยู่ในอาร์เรย์จะต้องเป็นข้อมูลชนิดเดียวกันเท่านั้น เช่น ถ้าเป็นอาร์เรย์ชนิดจำนวนเต็ม ข้อมูลทุกตัว ในอาร์เรย์ก็ต้องเป็นชนิดจำนวนเต็ม ไม่สามารถเก็บ ข้อมูลต่างชนิดกันได้



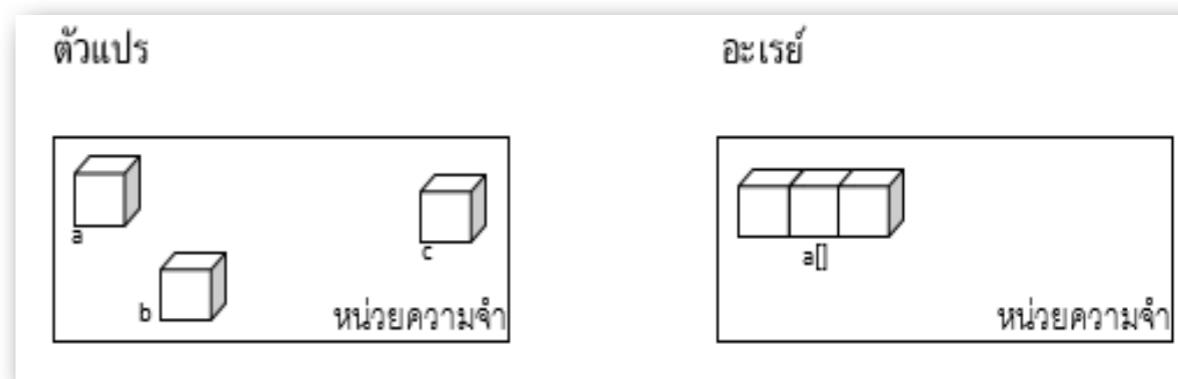
หากในการเขียนโปรแกรมต้องการตัวแปร ตัวมาเก็บค่าจำนวนเต็ม จำนวน เราสามารถประกาศตัวแปรเป็น `int x;` โดยที่ ตัวแปรตัวนี้จะเก็บค่าได้เพียง ค่าเท่านั้น ซึ่งหากต้องการตัวแปรมาเก็บจำนวนเต็ม 50 จำนวนจะต้องทำการประกาศตัวแปรเลย 50 ตัว เช่น `int x1, x2, x3, x4, x5.....X50;` ซึ่งเป็นการยุ่งยากในการเขียนโปรแกรมถ้าหากตัวแปร ตัวเปรียบเสมือนกล่องๆหนึ่ง โดยที่กล่องนี้จะมีขนาดเล็กหรือใหญ่นั้นจะขึ้นอยู่กับชนิดของข้อมูลหรือ *Data Type* ของตัวแปร และมีเงื่อนไขว่ากล่องหนึ่งสามารถเก็บค่าได้เพียงค่าเดียวเท่านั้น



ภาพแสดง Array ซึ่งเปรียบเสมือนกล่อง

ดังนั้นหากต้องการเก็บค่าใหม่จึงต้องกำหนดค่าเดิมทิ้งก่อน แต่ตัวแปรอาร์เรย์นั้นเปรียบเสมือนการนำกล่องมาเรียงต่อกันเป็นชุด ทำให้เก็บค่าได้จำนวนมาก โดยในการเก็บหรือการเข้าถึงนั้นต้องอ้างอิงจากลำดับของกล่อง

นอกจากนั้นแล้วจะเรียกแต่กต่างจากตัวแปรเรื่องตัวแหน่งที่อยู่ หากสร้างตัวแปรมา 3 ตัว คือ `int a, b, c;` ตัวแปรทั้ง 3 จะอยู่กระჯัดกระจายซึ่งอาจไม่ได้เรียงติดกัน ในหน่วยความจำ แต่ถ้าเป็นตัวแปรอะเรย์ `int a[] = new int[3];` แต่ละกล่องของอะเรย์ `a` นี้จะเรียงติดกัน



ภาพแสดงการเก็บ Array ในหน่วยความจำ

Section 1

อาร์เรย์ 1 มิติ (One Dimension Array)

มีการจัดเก็บข้อมูลในลักษณะต่อเนื่องกันเป็นแท่ง ซึ่งจะนำเสนอด้วยมุมมองแบบแนวตั้งแล้ววนอนก็ได้สัญลักษณ์ที่ใช้คือ `array_name[size]`

รูปแบบทั่วไปของโครงสร้างข้อมูลอาร์เรย์ 1 มิติ

รูปแบบ `ArrayName [L:U]`

โดย `ArrayName` คือ ชื่อของอาร์เรย์

L คือ ขอบเขตล่างสุด (Lower Bound)

U คือ ขอบเขตบนสุด (Upper Bound)

การประกาศอาร์เรย์ 1 มิติ ในภาษา Visual Basic

`Dim grades (5) as integer` เป็นการประกาศตัวแปรแบบอาร์เรย์ 1 มิติชื่อ `grade` ให้เป็นข้อมูลแบบตัวเลข โดยมีขนาดเท่ากับ 5 อีลีเมนต์ ซึ่งการประกาศตัวแปรนี้จะจองหน่วยความจำเท่ากับ $2 \text{ byte} * 5 = 10 \text{ byte}$

`Dim name (5) as String` เป็นการประกาศตัวแปรแบบอาร์เรย์ 1 มิติชื่อ `name` ให้เป็นข้อมูลแบบอักษร โดยมีขนาด

เท่ากับ 5 อีลีเมนต์ ซึ่งการประกาศตัวแปรนี้จะจองหน่วยความจำเท่ากับ $2 \text{ byte} * 5 = 10 \text{ byte}$

อาร์เรย์ `grades`

A	B	C	A	F
0	1	2	3	4

อาร์เรย์ `name`

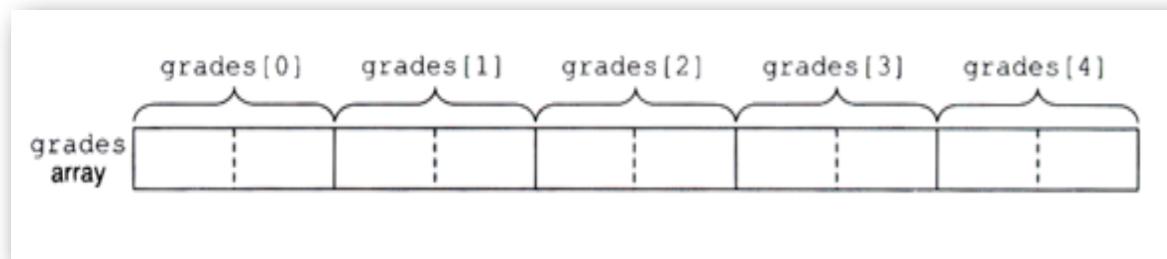
"Sam"
"Tony"
"David"
"Tum"
"Ton"
"Yod"

ในการเข้าถึงแต่ละอีลีเมนต์สามารถทำได้โดยการระบุค่าดัชนี (Index)

`grades(0)` อ้างถึงค่า `grades` แรกที่เก็บใน `grades array` คือ A

`grades(4)` อ้างถึงค่า `grades` ลำดับที่ห้าที่เก็บใน `grades array` F

`name(2)` อ้างถึงค่า `name` ลำดับที่สามที่เก็บใน David



ภาพแสดงการเก็บข้อมูล ในลักษณะของ Array

ในการอ้างอิงข้อมูลในอาร์เรย์ **ค่าดัชนีที่ใช้ในการอ้างอิงจะเริ่มต้นที่ 0** เสมอทั้งนี้เพื่อให้การเข้าถึงข้อมูลมีความรวดเร็ว ขึ้น จากรูป จะเห็นว่าค่าดัชนีจะเป็นตัวอ้างอิงถึงข้อมูลแต่ละอีลิเมนต์การเข้าถึงข้อมูลสามารถทำได้โดยการเลื่อนตำแหน่งของค่าดัชนีเพื่อไปยังอีลิเมนต์ต่างๆ ที่ต้องการ โดยเริ่มต้องจากตำแหน่งแรก จากตัวอย่างจะเป็นการเลื่อนตำแหน่งไปที่อีลิเมนต์ตัวที่ 3 นั่นเอง

ตัวแปรอาร์เรย์สามารถใช้งานได้เหมือนกับตัวแปรทั่วไป ตัวอย่างการใช้งานตัวแปร `grades` ที่เป็นอาร์เรย์ของจำนวนเต็ม 5 ตัว เช่น

```
grades (0) = 98;  
grades (i) = grades(0) - 11;  
grades (2) = 2 * (grades(0) - 6);  
grades (3) = 79;  
grades (4) = (grades(2) + grades(3) - 3 / 2;  
total = grades(0) + grades(1) + grades(2) + grades(3)
```

ค่าตัวเลขอีลิเมนต์ไม่จำเป็นต้องเป็นตัวเลขโดยตรง อาจเป็นตัวแปร หรือพจน์ของการกระทำที่ได้เป็นจำนวนเต็มก็ได้ เช่น

```
grades(i)  
grades(2 * i)  
grades(j - i)
```

ประโยชน์ที่เห็นได้ชัดอีกอย่างของการใช้พจน์ของจำนวนเต็มแทนการกำหนดค่าคงที่คือการใช้ร่วมกับการวนลูป for เช่น

```
total = grades(1) + grades(2) + grades(3) + grades(4)+  
grades(5)
```

เปลี่ยนเป็น

```

total = 0;
for (i=0;i <=4, ++i)
total += grades(i);

```

จะเห็นว่าจะมีการเก็บค่าใน `grades` แต่ละอีลิเม้นต์ตามลำดับเช่นกัน และสามารถใช้ `i` ซึ่งใช้เป็นตัวนับของลูป `for` ในขณะเดียวกันก็เป็นตัวชี้ให้กับอาร์เรย์ด้วย จะเห็นว่าจะมีประโยชน์มาก เมื่อมีการกระทำที่มีจำนวนครั้งมากกว่านี้

การกำหนดค่าให้อาร์เรย์

```

Dim intArray() as Integer = {9 , 5, 2, -3, 17, 9, 0}
Dim strArray(2) as String
strArray(0) = "ชมพู"
strArray(1) = "ชมจันทร์"
strArray(2) = "ณเดช"

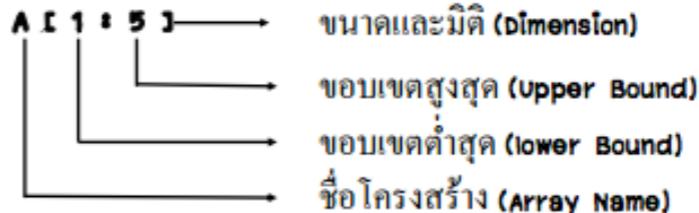
```

การกำหนดค่าเริ่มต้นของตัวแปรแบบอาร์เรย์ 1 มิติ สามารถกำหนดได้สำหรับทุกชนิดข้อมูล ดังตัวอย่างต่อไปนี้

ขอบเขตของอาร์เรย์ 1 มิติ

เลขด้วยในอาร์เรย์ประกอบด้วยช่วงขอบเขตของค่าซึ่งประกอบด้วยขอบล่างสุด (Lower Bound) และขอบเขตบนสุด (Upper Bound)

A[1:5]				
DATA	DATA	DATA	DATA	DATA



ภาพแสดงขอบเขตบนสุด และขอบเขตล่างสุดของ Array

การคำนวณหาจำนวนสมาชิกของอาร์เรย์หนึ่งมิติ

$$\text{จำนวนสมาชิก} = U-L+1$$

เช่น `num[1:5]` มีสมาชิก เท่ากับ $5-1+1 = 5$ ตัว

การคำนวณหาตำแหน่ง(Address) ในหน่วยความจำของอาร์เรย์หนึ่งมิติ

$$(A[i]) = B + w * (i - L)$$

โดยที่

i = ตำแหน่งที่ต้องการ

B = ตำแหน่งเริ่มต้น ในหน่วยความจำ

L = ตำแหน่งเริ่มต้น

w = ขนาดความกว้างของชนิดข้อมูลของข้อมูล(byte)

ตัวอย่างที่ 1

○ กำหนดให้ $B = 1000$

$w = 1$ ไบต์

อนุญาตทราบว่า อาร์เรย์ $a[10]$ ถูกจัดเก็บ

ไว้ในช่องของความจำของตัวแปร a

จากสูตร

$$LOC(a[i]) = B + w(i-L)$$

$$LOC(a[10]) = 1000 + 1(10 - 0)$$

$$= 1010$$

ดังนั้นค่าหนึ่งของอาร์เรย์ $a[10]$ จะถูกเก็บไว้

ในหน่วยความจำแอดเดรสที่ 1010

Base Add	Memory
1000	$a[0]$
1001	$a[1]$
	.
1010	$a[10]$
	.
n	$a[n]$

ตัวอย่างที่ 2

ต้องการประมวลผลอาเรย์ income เพื่อ

จัดเก็บข้อมูลรายได้ของปี พ.ศ. 2541-2550 ซึ่ง

เข้าไปพิมพ์ข้อมูล income[2541:2550]

กำหนดให้ Base Address (B) = 7000

$w = 4$ ไบต์

จึงคำนวณหาผลโดยใช้ตัวแปร i จัดเก็บข้อมูลรายได้ของปี

2549

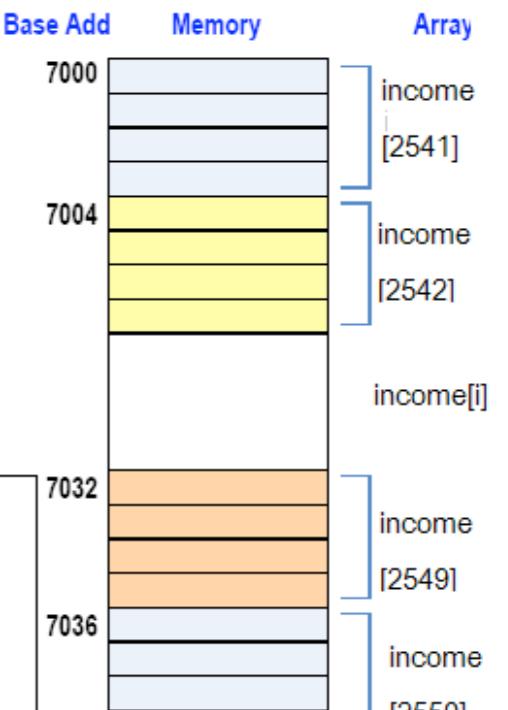
$$LOC(income[i]) = B + w(i-L)$$

$$LOC(income[2549]) = 7000 + 4(2549 - 2541)$$

$$= 7032$$

ดังนั้นข้อมูลรายได้ของปี 2549 จะถูกจัดเก็บไว้

ในหน่วยความจำแอดเดรส 7032



Section 2

อาร์เรย์ 2 มิติ

โครงสร้างข้อมูลที่มีการจัดเก็บข้อมูลแบบตารางสองทาง ข้อมูลมีการจัดเรียงกันตามแนว **แถว (Row)** และ **คอลัมน์ (Column)** การอ้างถึงข้อมูลต้องระบุตำแหน่งแถวและตำแหน่งหลักที่ข้อมูลนั้นอยู่

Columns			
Rows	0	1	2
0	10	5	3
1	2	1	9
2	11	6	7
3	0	4	3

ภาพแสดงลักษณะของอาร์เรย์ 2 มิติ

รูปแบบทั่วไปของโครงสร้างข้อมูลอาร์เรย์ 2 มิติ

ArrayName[L₁ : U₁, L₂ : U₂]

เมื่อ ArrayName คือ ชื่อของโครงสร้างข้อมูลอาร์เรย์

L₁ คือ ค่าขอบเขตล่างสุด (Lower Bound) ของแถว

U₁ คือ ค่าขอบเขตสูงสุด (Upper Bound) ของแถว

L₂ คือ ค่าขอบเขตล่างสุด (Lower Bound) ของคอลัมน์

U₂ คือ ค่าขอบเขตสูงสุด (Upper Bound) ของคอลัมน์

การประกาศอาร์เรย์ 2 มิติ ในภาษา Visual Basic

Dim ourArray (1, 9) as integer เป็นการประกาศตัวแปรแบบอาร์เรย์ 2 มิติชื่อ ourArray ซึ่งประกอบไปด้วย 2 แถว 10 คอลัมน์ จะเก็บข้อมูลได้ทั้งหมด 20 อีลีเมนต์

การอ้างอิงถึงข้อมูลสำหรับอีลีเมนต์ใดในอาร์เรย์ 2 มิติ ทำได้โดยการระบุค่าเลขด้วยชื่อทั้ง 2 แนว คือ แนวแถวและแนวคอลัมน์

จากรูป ตำแหน่งของ ourArray(1 , 7) จะสามารถบอกตำแหน่ง กำหนดค่าดังนี้ในแนวแคลเป็น 2 และในแนวคอลัมน์ 8 เช่นเดียวกับตัวแปรอาร์เรย์ 1 มิติ และตัวแปรทั่วไป

การกำหนดค่าให้ Array 2 มิติ

```
Dim gradeStd(3 , 2) as Integer
    gradeStd (0,0) = 50          ให้แคลที่ 1 คอลัมน์ที่ 1 เก็บข้อมูล 50
    gradeStd (0 , 1) = 20        ให้แคลที่ 1 คอลัมน์ที่ 2 เก็บข้อมูล 20
    gradeStd (0 , 2) = 30        ให้แคลที่ 1 คอลัมน์ที่ 2 เก็บข้อมูล 30
```

การคำนวณหาจำนวนสมาชิกของอาร์เรย์สองมิติ

$$\text{จำนวนสมาชิก} = (U_1 - L_1 + 1) * (U_2 - L_2 + 1)$$

โดย U_1 = ขอบเขตบนสุด ของแคล

L_1 = ขอบเขตล่างสุด ของแคล

U_2 = ขอบเขตบนสุด ของคอลัมน์

L_2 = ขอบเขตล่างสุด ของคอลัมน์

	c0	c1	c2	c3
r0				
r1				
r2				
r3				
r4				

row column

Int number [5][4];

$$\text{จำนวนสมาชิก} = (U_1 - L_1 + 1) * (U_2 - L_2 + 1)$$

$$= (4-0+1) * (3-0+1)$$

$$= 5 * 4$$

$$= 20$$

การคำนวณหาจำนวนสมาชิกของอาร์เรย์สองมิติ

$$\text{จำนวนสมาชิก} = (U_1 - L_1 + 1) * (U_2 - L_2 + 1)$$

โดย U_1 = ขอบเขตบนสุด ของแคล

L_1 = ขอบเขตล่างสุด ของแคล

U_2 = ขอบเขตบนสุด ของคอลัมน์

L_2 = ขอบเขตล่างสุด ของคอลัมน์

การจัดเก็บอาร์เรย์สองมิติในหน่วยความจำ จะมี 2 แบบ

1. การจัดเก็บด้วยการเรียงແລວเป็นหลัก (Row Major Order) สูตรการคำนวณหาตำแหน่งที่ใช้เก็บข้อมูลในอาร์เรย์

$$LOC(K[i,j]) = B + w[C(i-L_1) + (j-L_2)]$$

โดยที่

$LOC(K[i,j])$ = ตำแหน่งแอดเดรสที่เก็บ $K[i,j]$ ในหน่วยความจำ

B = แอดเดรสเริ่มต้น (Base Address)

w = จำนวนช่องของหน่วยความจำที่จัดเก็บข้อมูลต่อหนึ่งสมາชิก

i = ตำแหน่งของແລວในอาร์เรย์

j = ตำแหน่งของคอลัมน์ในอาร์เรย์

L_1 = ค่าขอบเขตล่างสุด (lower Bound) ของคอลัมน์

L_2 = ค่าขอบเขตล่างสุด (lower Bound) ของคอลัมน์

C = จำนวนคอลัมน์ของແລວลำดับ

ตัวอย่าง ต้องการทราบตำแหน่งแอดเดรสที่เก็บข้อมูลอาร์เรย์ K แถวที่ 2 คอลัมน์ 1 ($K[2,1]$) กำหนดให้ $B = 500$ $W = 4$ ไบต์

		Columns		
		0	1	2
Rows	0	$K[0, 0]$	$K[0, 1]$	$K[0, 2]$
	1	$K[1, 0]$	$K[1, 1]$	$K[1, 2]$
	2	$K[2, 0]$	$K[2, 1]$	$K[2, 2]$
	3	$K[3, 0]$	$K[3, 1]$	$K[3, 2]$

$K[0:3, 0:2]$

การคำนวณ

สูตร

$$LOC(K[i,j]) = B + w[C(i-L_1) + (j-L_2)]$$

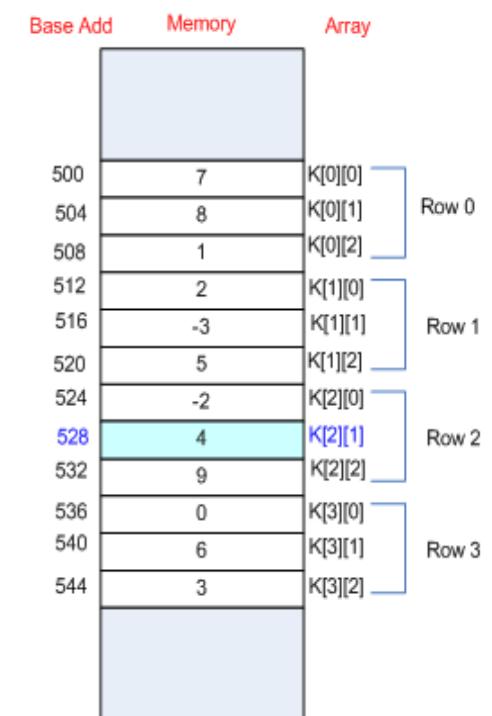
$$LOC(K[2,1]) = 500 + 4[3(2-0) + (1-0)]$$

$$= 500 + 4[6+1]$$

$$= 500 + 28$$

$$= 528$$

ดังนี้ อาร์เรย์ K แถวที่ 2 คอลัมน์ 1 จะจัดเก็บอยู่ในตำแหน่ง แอดเดรสที่ 528



2. การจัดเก็บด้วยการเรียงคอลัมน์เป็นหลัก (Column Major Order)

$$LOC(K[i,j]) = B + w[R(j-L_2) + (i-L_1)]$$

โดยที่

$LOC(K[i,j])$ = ตำแหน่งแอดเดรสที่เก็บ $K[i,j]$ ในหน่วยความจำ

B = แอดเดรสเริ่มต้น (Base Address)

w = จำนวนช่องของหน่วยความจำที่จัดเก็บข้อมูลต่อหนึ่งสามาชิก

l = ตำแหน่งของแطاในอาร์เรย์

j = ตำแหน่งของคอลัมน์ในอาร์เรย์

L_1 = ค่าขอบเขตล่างสุด (lower Bound) ของคอลัมน์

L_2 = ค่าขอบเขตล่างสุด (lower Bound) ของคอลัมน์

R = จำนวนแطاของแطاลำดับ

ตัวอย่าง

		Columns		
		0	1	2
Rows	0	K[0,0]	K[0,1]	K[0,2]
	1	K[1,0]	K[1,1]	K[1,2]
	2	K[2,0]	K[2,1]	K[2,2]
	3	K[3,0]	K[3,1]	K[3,2]

K[0:3 , 0:2]

7	8	1
2	-3	5
-2	4	9
0	6	3

ต้องการทราบตำแหน่งของออดเดรสที่เก็บข้อมูลอาร์เรย์ K แถวที่ 2 คอลัมน์ที่ 1 โดยการจัดเรียงข้อมูลให้แน่นอย่างความจำเพาะก็ต้องรู้ชั้นของคอลัมน์เป็นหลัก
กำหนดให้ $B = 500$

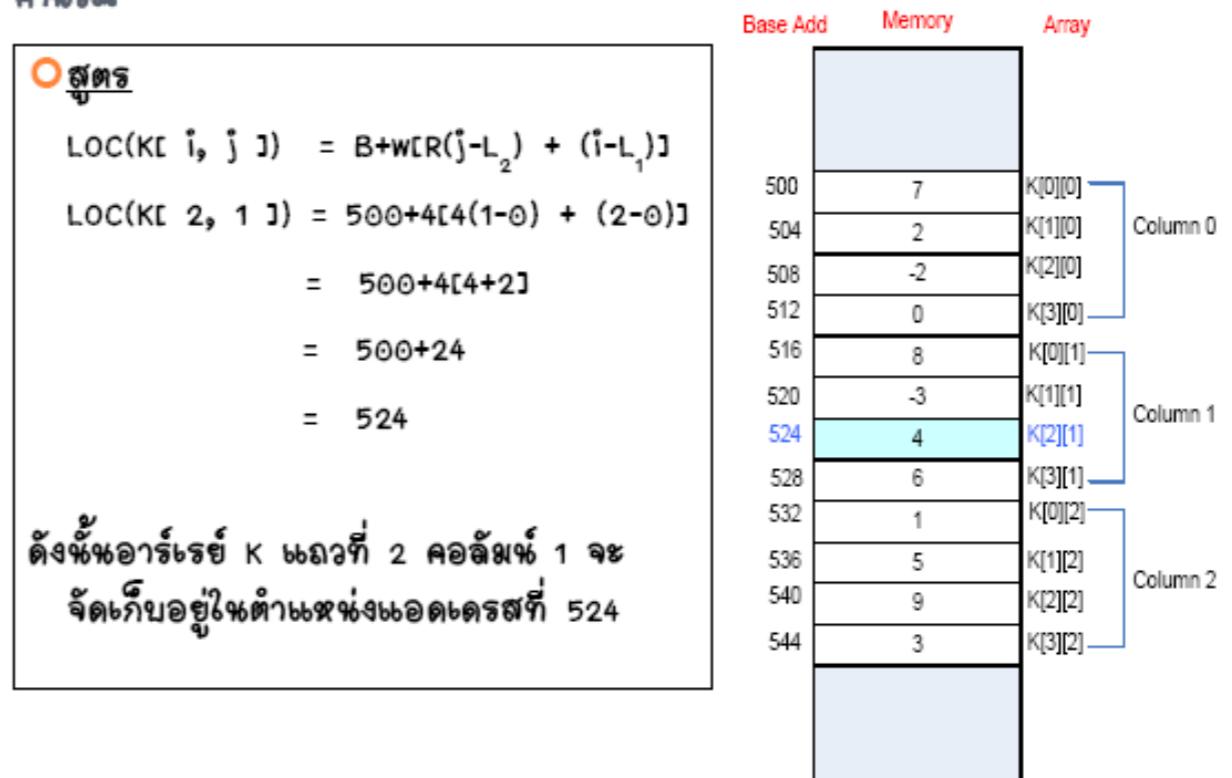
คำนวณ

● ลูป

$$LOC(K[i, j]) = B + w[R(j-L_2) + (i-L_1)]$$

$$\begin{aligned} LOC(K[2, 1]) &= 500 + 4[4(1-0) + (2-0)] \\ &= 500 + 4[4+2] \\ &= 500 + 24 \\ &= 524 \end{aligned}$$

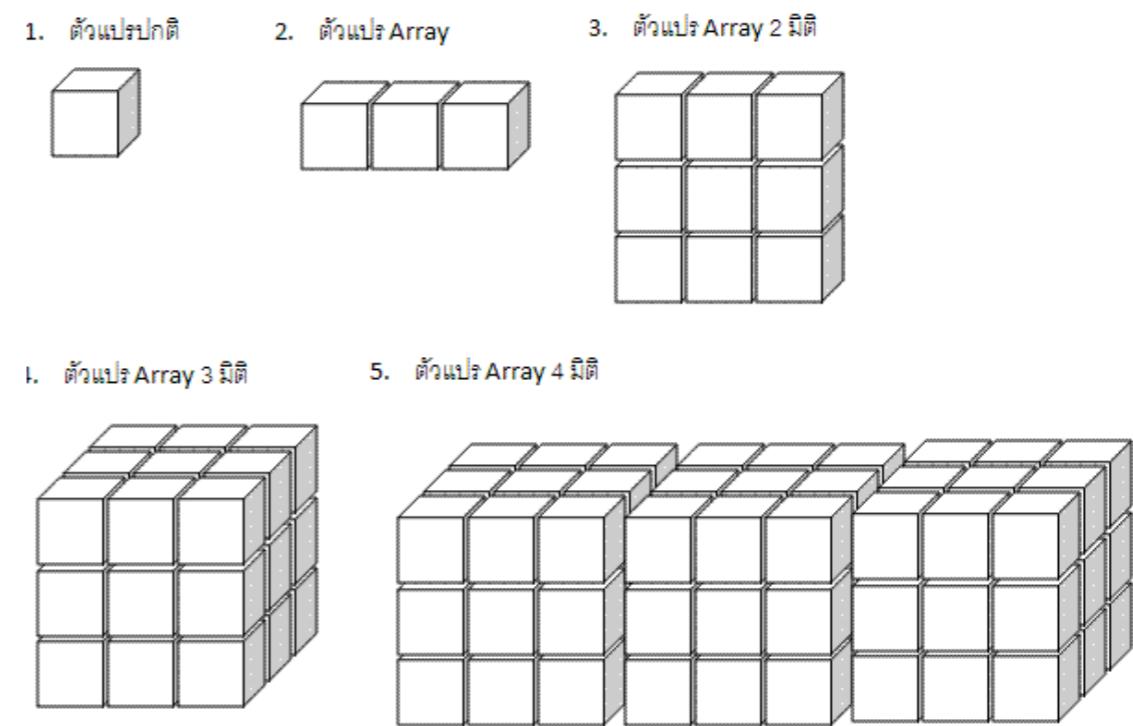
ดังนี้ อาร์เรย์ K แถวที่ 2 คอลัมน์ 1 จะจัดเก็บอยู่ในตำแหน่งของออดเดรสที่ 524



Section 3

อาร์เรย์ 3 มิติ

อาร์เรย์สามมิติคือการนำเอาอาร์เรย์สองมิติมาเรียงช้อนกันหลายๆชั้น (page) ดังนั้นจึงทำให้อาร์เรย์สามมิติ จะมีลักษณะเป็น~~ແດວ~~และ~~ຄອລັມນີ້~~ແລ້ວກ็จะมี**ความลົກເພີ່ມຂຶ້ນ**มา



ภาพแสดงอาร์เรย์ในมิติต่าง ๆ

รูปแบบทั่วไปของโครงสร้างข้อมูลอาร์เรย์ 3 มิติ

ArrayName [L₁ : U₁ , L₂ : U₂ , L₃ : U₃]

เมื่อ ArrayName คือ ชื่อของโครงสร้างข้อมูลอาร์เรย์

L₁ คือ ค่าขอบเขตล่างสุด (Lower Bound) ของแถว

U₁ คือ ค่าขอบเขตสูงสุด (Upper Bound) ของแถว

L₂ คือ ค่าขอบเขตล่างสุด (Lower Bound) ของคอลัมน์

U₂ คือ ค่าขอบเขตสูงสุด (Upper Bound) ของคอลัมน์

L₃ คือ ค่าขอบเขตล่างสุด (Lower Bound) ของความลึก

U₃ คือ ค่าขอบเขตสูงสุด (Upper Bound) ของความลึก

การหาจำนวนสมาชิกของอาร์เรย์ 3 มิติ

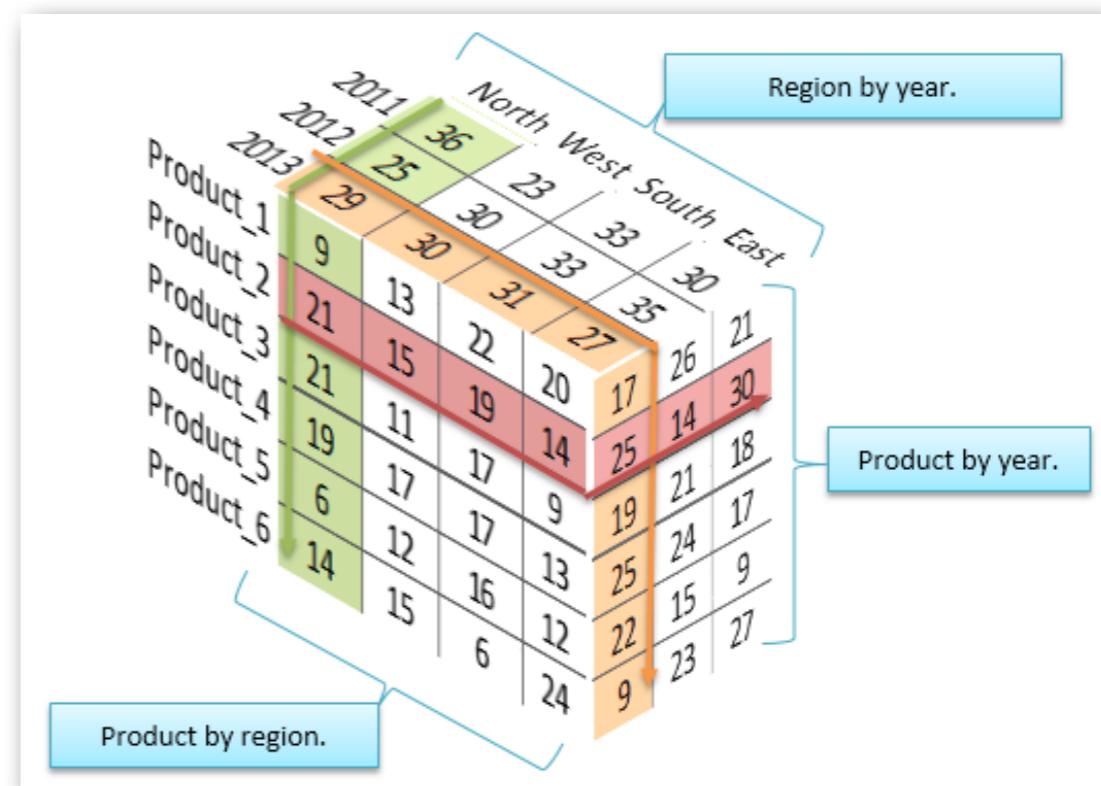
$$\text{จำนวนสมาชิก} = (U_1 - L_1 + 1) * (U_2 - L_2 + 1) * (U_3 - L_3 + 1)$$

ตัวอย่าง จำนวนสมาชิกของอาร์เรย์ A(2,3,4) หรือ
A(0:1,0:2,0:3)

$$\text{จำนวนสมาชิก} = (1-0+1)*(2-0+1)*(3-0+1)$$

การประกาศอาร์เรย์ 3 มิติ ในภาษา Visual Basic

Dim theirArray (3 , 4 , 2) as Single เป็นการประกาศตัวแปรแบบอาร์เรย์ 3 มิติชื่อ theirArray เก็บทศนิยมได้ 60 ตัว



ภาพแสดงการจัดเก็บข้อมูลในลักษณะอาร์เรย์ 3 มิติ

การคำนวณหาแอดเดรสของอาร์เรย์สามมิติแบบแคล เป็นหลัก

$$LOC(S[p,i,j]) = B + [W * R * C(p - L_1)] + [W * C(i - L_2)] + [W(j - L_3)]$$

โดยที่

$LOC(S[p,i,j])$ = ตำแหน่งแอดเดรสที่เก็บ $S[p,i,j]$ ในหน่วยความจำ

B = แอดเดรสเริ่มต้น

W = จำนวนช่องของหน่วยความจำที่จัดเก็บข้อมูล

P = ตำแหน่งของชั้นในอาร์เรย์

i = ตำแหน่งของแคล ในอาร์เรย์

j = ตำแหน่งของคอลัมน์ในอาร์เรย์

R = จำนวนแคลของแคลลำดับ

C = จำนวนคอลัมน์ของแคลลำดับ

L_1 = ค่าขอบเขตล่างสุด (lower Bound) ของชั้น

L_2 = ค่าขอบเขตล่างสุด (lower Bound) ของแคล

L_3 = ค่าขอบเขตล่างสุด (lower Bound) ของคอลัมน์

ตัวอย่าง

- ต้องการทราบตำแหน่งแอดเดรสที่เก็บข้อมูลอาร์เรย์ S ชั้นที่ 0 แถวที่ 3 คอลัมน์ที่ 4

กำหนดให้ $B=500$
 $W=4$ ไบต์

int $S[3][4][5];$

ถูตร $LOC(S[i,j,k]) = B + [W * R * C(p - L_1)] + [W * C(i - L_2)] + [W(j - L_3)]$

$$LOC(S[0,3,4]) = 500 + [4 * 4 * 5(0 - 0)]$$

$$+ [4 * 5(3 - 0)] + [4(4 - 0)]$$

$$= 500 + 0 + 60 + 16$$

$$= 576$$

ตารางแสดง S ที่เก็บข้อมูลในรูปแบบที่เข้ามาให้



Section 4

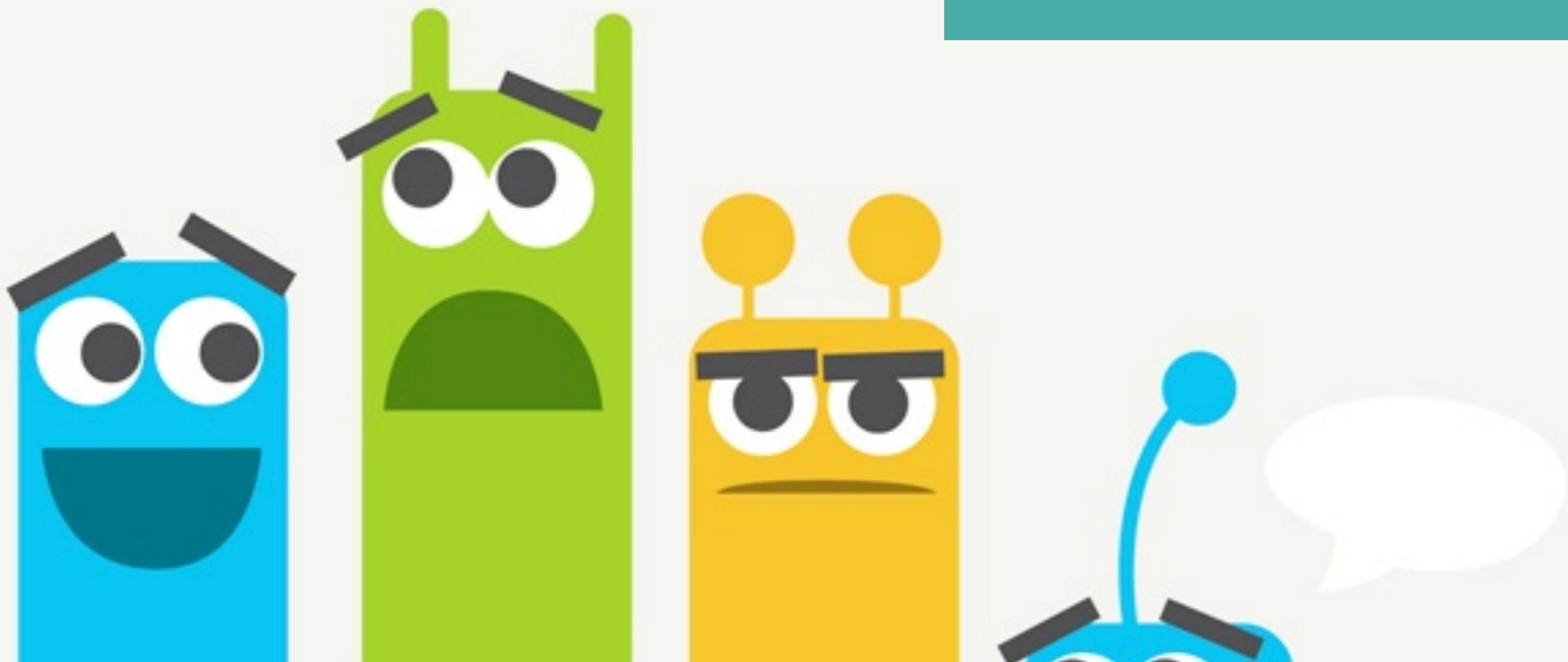
แบบฝึกหัด

1. อาร์เรย์ $A[0 : 2, 1 : 3]$ มีจำนวนช่องของอาร์เรย์กี่ช่อง วัดภาพอาร์เรย์ประกอบด้วย
2. จากอาร์เรย์ $A[0 : 2, 1 : 3]$ ถ้าอยากรابข้อมูลของ แถวที่ 1 คอลัมน์ที่ 2 จะเขียนในลักษณะอาร์เรย์ย่างไรในการอ้างอิง index
เช่น score [10] คะแนนของคนที่ 10 หรือแถวที่ 10
3. กำหนดให้อาร์เรย์ $A[0:4, 0: 27]$ ของพื้นที่หน่วยความจำโดยสมาชิกแต่ละตัว ใช้เนื้อที่ 4 ไบต์ และ $A[4]$ ถูกบันทึกที่ตำแหน่งที่ 300 จากโจทย์จงหาตำแหน่งของ $A[5, 23]$
4. ประกาศตัวแปรอาร์เรย์ $x[4:6]$ กำหนดค่าแอ็ดเดรสเริ่มต้น 200 และใช้เนื้อที่เก็บข้อมูลเท่ากับ 4 ไบต์ อยากรابว่า $x[5]$ อยู่ตำแหน่งแอ็ดเดรสใดในหน่วยความจำ
5. Array ขนาด 3 มิติ $A(2:8, 10:19, 20:29)$ โดย $A(2, 10, 20)$ อยู่ที่ 3722 แต่ละช่องมีขนาด 32 bit จงหาจำนวนช่องทั้งหมด และหา address ของ $A(3, 15, 22)$

6. จงอธิบายจุดเด่นของ โครงสร้างข้อมูลแบบอาร์เรย์มีอะไรบ้าง

7. จงเขียนอัลกอริธึมอ่านค่าตัวเลขจำนวน 10 ค่าเพื่อเก็บไว้ในตัวแปรอาร์เรย์ จากนั้นให้ทำการค้นหาค่าสูงสุดในอาร์เรย์

Link list Structure



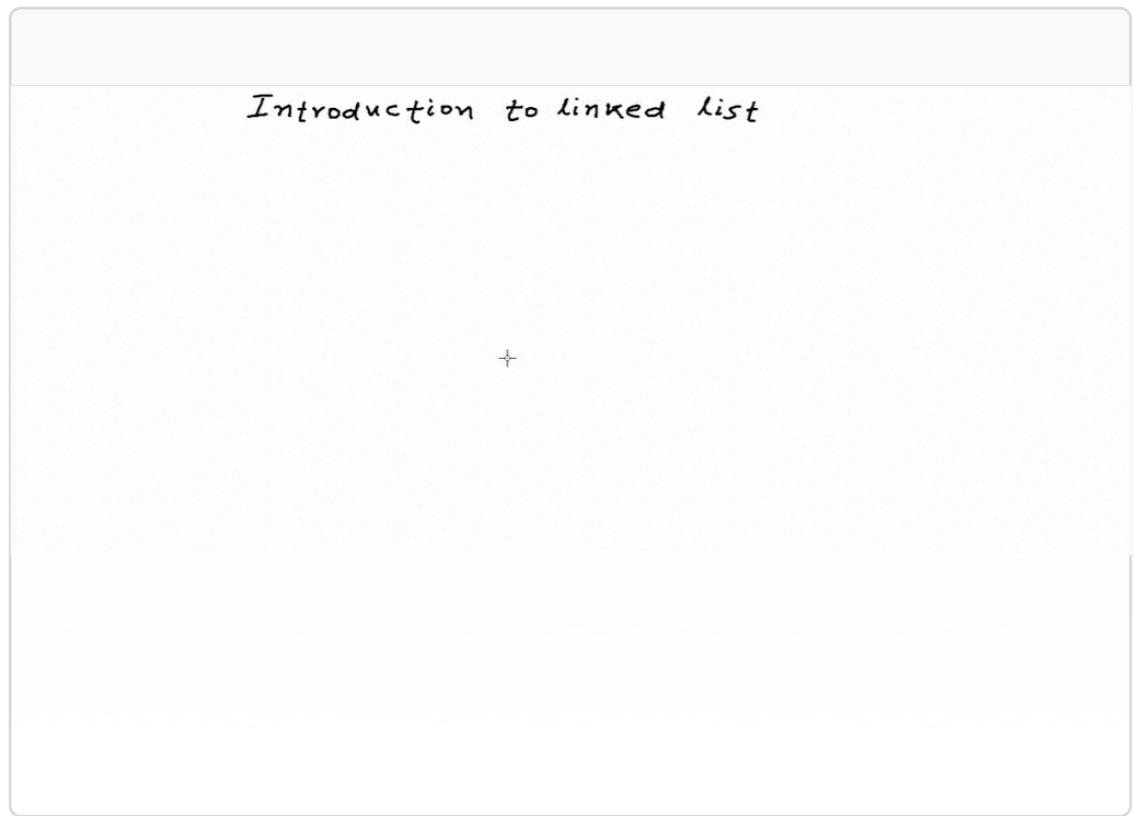
วัตถุประสงค์

- 1.เข้าใจหลักการและแนวคิดของลิสต์แบบเชิงเส้น
- 2.อธิบายการดำเนินงานพื้นฐานของลิสต์ได้
- 3.อธิบายโครงสร้างข้อมูลแบบลิงค์ลิสต์ได้

Section 1

นิยามโครงสร้างข้อมูลลิงค์ลิสต์ (Link list)

Movie 4.1 แนะนำ Link list เป็นต้น



ที่มา : <http://www.youtube.com/watch?v=NobH1GUjV3g>

โครงสร้างข้อมูลแบบลิงค์ลิสต์ ถือเป็นโครงสร้างข้อมูลที่สำคัญมาก อีกประเภทหนึ่ง เพราะเป็นโครงสร้างข้อมูลที่สร้างขึ้นมาเพื่อหลีกเลี่ยง ปัญหาการเสียเวลาในการเพิ่มและลบข้อมูลของโครงสร้างข้อมูลแบบ อาร์เรย์

ปัญหาและความยุ่งยากในการปฏิบัติการเพิ่มข้อมูลเข้าหรือลบ ข้อมูลออกจากจากอาร์เรย์ อาจแก้ไขได้โดยการใช้ลิงค์ลิสต์ ซึ่งข้อมูล แต่ละตัวถูกจัดเก็บอย่างกระจัดกระจาย ในหน่วยความจำหลัก **อาจจะ** **ไม่ได้ใช้เนื้อที่ที่เรียงต่อเนื่องกัน** แต่มีการเชื่อมโยงถึงกันระหว่าง ข้อมูลได้โดยใช้ตัวชี้ภายใน **โหนด** (Internal Pointer) หรือเรียกอีก อย่างว่า **ลิงค์ฟิลด์** (Linked Field) **เก็บเลขที่ความจำหลัก** (address) ของข้อมูลที่มาเชื่อมต่อกัน ซึ่งความสัมพันธ์ของข้อมูลแต่ละตัวยังคง เรียงลำดับกันในมโนภาพของนักเขียนโปรแกรม แต่ตำแหน่งที่อยู่ ในหน่วยความจำหลักไม่จำเป็นต้องเรียงลำดับกัน และระหว่างข้อมูล ใดๆ ในลิงค์ลิสต์อาจมีเนื้อที่ว่างคันอยู่ได้เสมอ ข้อมูลทั้งหมดในลิงค์ลิสต์จะเชื่อมต่อกันด้วยค่าในลิงค์ฟิลด์ นอกจากนี้ต้องมีตัวชี้ภายนอก **โหนด** (External Pointer) หรือเรียกสั้นๆ ว่า **ตัวชี้** (Pointer) ที่ทำ หน้าที่เก็บตำแหน่งของข้อมูลแรก ในลิงค์ลิสต์ด้วย เพื่อเป็นจุดเริ่มต้น ในการเข้าถึงข้อมูลต่างๆ

การแทนที่ลิงค์ลิสต์ในความจำหลัก

การแทนที่ลิงค์ลิสต์ในหน่วยความจำหลักสามารถกระทำได้ 2 แบบคือ การแทนที่ลิงค์ลิสต์แบบไนามิกและการแทนที่ลิงค์ลิสต์แบบสแตติก

1. **การแทนที่ลิงค์ลิสต์แบบไนามิก** เป็นการนำชนิดของข้อมูลแบบ ตัวชี้ หรือ พอยเตอร์ มาประยุกต์ในการสร้างลิงค์ลิสต์ซึ่งภาษาคอมพิวเตอร์ที่มีความสามารถในการจัดการหน่วยความจำแบบไนามิกได้แก่ ภาษาปาสคาล ภาษาซี เป็นต้น การจัดการหน่วยความจำประเภทนี้มีความยืดหยุ่นสูง ผู้เขียนโปรแกรมไม่ต้องจอง (allocate) เนื้อที่หน่วยความจำไว้ล่วงหน้า สามารถจองหรือปล่อย (free) หน่วยความจำได้ขณะที่โปรแกรมกำลังทำงาน

ในการใช้งานหน่วยความจำแบบไนามิกนั้น จะใช้หน่วยความจำในส่วนของสตอร์เรจพูล (storage pool) ซึ่งเป็นเนื้อที่ส่วนหนึ่งในหน่วยความจำหลัก โดยภายในสตอร์เรจพูลจะเก็บโหนดอิสระ (free node) ที่ยังไม่ได้ถูกนำไปใช้งาน ทุกครั้งที่มีการเพิ่มโหนดใหม่เข้าสู่ลิสต์ที่กำลังทำงานอยู่ ต้องขอโหนดอิสระจากสตอร์เรจพูลเสมอ

สำหรับงานที่มีการเพิ่มโหนดเข้าและดึงโหนดออกจากลิสต์ค่อนข้างบ่อย โหนดที่ถูกดึงออกจากลิสต์แล้วควรจะต้องนำมาระบุกลับคืนในสตอร์เรจพูลด้วย เพื่อให้สามารถนำ

กลับมาใช้งานใหม่ได้ วิธีการนี้เรียกว่า การเก็บขยะ (garbage collection) มีฉะนั้นแล้วเมื่อเราขอโหนดจากสตอร์เรจพูลบ่อยๆ และทำให้โหนดอิสระหมด อาจทำให้การปฏิบัติงานหยุดชะงักได้ กรณีนี้เรียกว่า หน่วยความจำไม่พอ (memory overflow)

2. **การแทนที่ลิงค์ลิสต์แบบสแตติก** โดยทั่วไปการสร้างลิงค์ลิสต์นิยมใช้ชนิดข้อมูลแบบ ตัวชี้ หรือพอยเตอร์ แต่ในภาษาคอมพิวเตอร์บางภาษาไม่มีชนิดข้อมูลแบบนี้ เช่น ภาษาโคงอล ภาษาฟอร์แทรน และภาษาเบลสิก เป็นต้น ดังนั้นเมื่อต้องการแทนที่ลิงค์ลิสต์ในหน่วยความจำหลักจะต้องใช้แล้วล้ำด้วย โดยต้องมีการจองเนื้อที่ไว้ในขนาดที่คงที่แน่นอน และสามารถใช้เนื้อที่ได้ไม่เกินที่ประกาศไว้ เป็นการแทนที่ลิงค์ลิสต์แบบสแตติก

Section 2

ชนิดของ Link list

โดยทั่วไปลักษณะของลิงค์ลิสต์ประกอบไปด้วยสมาชิกซึ่งจะถูกเรียกว่า โหนด (node) ที่อยู่กันอย่างเป็นลำดับ จำนวนสมาชิกในลิสต์อาจเปลี่ยนแปลงได้ รูปแบบการจัดโครงสร้างของลิสต์แบบเชื่อมโยงทำได้โดยการเพิ่มลิงค์ฟิลด์เพื่อเก็บที่อยู่ของโหนดข้างเคียง ลิงค์ลิสต์อาจมีลิงค์ฟิลด์ได้หลาย ๆ ฟิลด์แล้วแต่งานที่ทำ เราสามารถจำแนกลิงค์ลิสต์ออกเป็นชนิดต่างๆ ตามลักษณะและจำนวนของลิงค์ฟิลด์ได้ดังนี้

1. ลิงค์ลิสต์เดี่ยว (singly linked list) เป็นลิงค์ลิสต์ที่แต่ละโหนดมีเพียง 1 ลิงค์ฟิลด์ โดยในแต่ละโหนดจะประกอบไปด้วย 2 ส่วน คือ ส่วนของข้อมูลข่าวสาร (info) และส่วนของลิงค์ฟิลด์ (link) ดังภาพ

info link

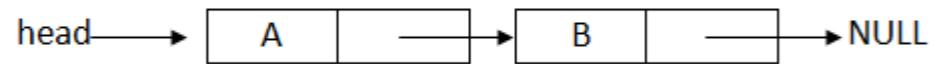
ส่วนที่เป็นข้อมูล (info)

ส่วนนี้เป็นส่วนที่ใช้เก็บข้อมูลต่างๆ ของแต่ละสมาชิกบนลิงค์ลิสต์ และข้อมูลที่ทำการเก็บนี้อาจจะเป็นกลุ่มของข้อมูลซึ่งมีข้อมูลหลายๆ ชนิด มาอยู่ด้วยกันก็ได้

ส่วนที่เป็นตัวชี้หรือลิงค์ฟิลด์

เป็นส่วนที่ใช้สำหรับเชื่อมโยงข้อมูลตัวถัดไป เปรียบเทียบได้กับมือที่คอยชี้ไปยังสมาชิกตัวที่อยู่ถัดจากตัวเองไป โดยลิงค์ฟิลด์จะเก็บค่าตำแหน่ง (address) ของสมาชิกตัวถัดไป จึงทำให้แต่ละสมาชิกของลิงค์ลิสต์ให้อยู่รวมกันเป็นสายข้อมูล ทำให้สามารถที่จะจัดการกับโครงสร้างข้อมูลประเภทนี้ได้มีประสิทธิภาพกว่า โครงสร้างแบบอาร์เรย์ เพราะการเพิ่มข้อมูลจะทำได้โดยการสร้างสมาชิกของลิงค์ลิสต์ขึ้นมาใหม่ได้ตามจำนวนที่ต้องการ โดยไม่มีการจำกัดจำนวน จำนวนนั้นจะทำการย้ายลิงค์ฟิลด์ให้ชี้ไปยัง โหนดที่ต้องการจะเชื่อม หากโหนดใดเป็น โหนดสุดท้ายส่วนที่เป็นตัวชี้จะเก็บข้อมูลที่เป็นค่าเฉพาะ เรียกว่า NULL หรือ NIL ซึ่งหมายถึงไม่ได้ชี้ไปยังโหนดใดๆ เป็นตัวบ่งบอกการสิ้นสุดของลิสต์

นอกจากสมาชิกซึ่งแต่ละตัวเรียกว่า โหนดแล้ว โครงสร้างข้อมูลประเภทนี้ยังต้องมี **ตัวชี้ภายนอก (External pointer)** เพื่อเก็บตำแหน่งเริ่มต้นของลิสต์ ซึ่งก็คือ โหนดแรกในลิสต์นั้นเอง โดยในเอกสารนี้จะแทนด้วยตัวแปรชื่อ head ดังภาพ



การสร้างลิงค์ลิสต์เดี่ยว

ในการสร้างลิงค์ลิสต์เดี่ยวขึ้นมาใช้งานในนั้น จะต้องประกาศโครงสร้างของโหนดและตัวชี้ภายในนอกดังนี้

```

typedef struct list {
    int      info;
    struct list*link;
} node;
node    *head, *temp, *current, *pred;

```

พิจารณาการประกาศตัวแปรข้างต้น แสดงว่า head, temp, current และ pred เป็นตัวแปรชนิดพอยเตอร์ที่ชี้ไปยังโครงสร้างที่ประกอบไปด้วย 2 พลัดย่อຍ คือ

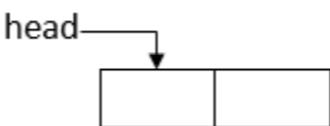
ฟลัดข้อมูล (Data Field) ได้แก่ info

ฟลัดตัวชี้ (Linked Field) ได้แก่ link

โดยฟลัดตัวชี้จะทำหน้าที่เก็บตำแหน่งของโครงสร้างถัดไป ดังนั้น ฟลัด link นี้เองที่ทำหน้าที่เชื่อมโยงทุกโครงสร้างเข้าด้วยกัน เมื่อประกาศโครงสร้างของโหนดเรียบร้อยแล้ว

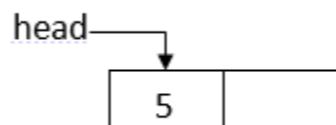
ในขั้นตอนต่อไปคือการสร้างโหนดแรกของลิสต์ ซึ่งมีกระบวนการดังนี้

ยึดโหนดว่างจาก Storage Pool โดยให้ตัวชี้ head ชี้ไปยังโหนดว่างนั้น ซึ่งทำได้โดยการใช้ประโยชน์ค NEW ดัง ตัวอย่าง



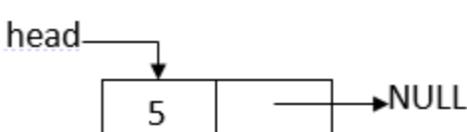
head = NEW node;

นำข้อมูลเข้าไปเก็บในโหนดใหม่นั้น



head -> info = 5;

กำหนดให้ลิงค์ฟลัดของโหนดใหม่นี้เก็บค่า NULL
เนื่องจากเป็นโหนดเดียวของลิสต์



```
head -> link = NULL;
```

ตัวอย่างฟังก์ชันที่ใช้ในการสร้างโหนดแรกของลิงค์ลิสต์เดี่ยว

```
void CreateSingly (void) {
```

```
    if ( head == NULL ) {
```

```
        head = NEW node;
```

```
        head -> info = X;
```

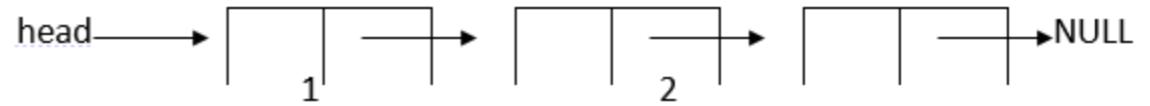
```
        head -> link = NULL;
```

```
}
```

```
}
```

การแสดงข้อมูลในลิงค์ลิสต์เดี่ยว

เป็นการเข้าไปในลิสต์เพื่อแสดงข้อมูลในแต่ละโหนด ถ้าต้องการแสดงรายละเอียดข้อมูลของทุกโหนดก็จะต้องเข้าไปในลิสต์โดยเริ่มจากโหนดแรก และไปยังโหนดถัดไปโดยตามตำแหน่ง ในลิงค์ไปจนกระทั่งครบทุกโหนด ซึ่งก็คือ ลิงค์มีค่าเป็น NULL เราไม่สามารถไปยังสมาชิกตัวที่ n ไดๆ ในลิงค์ลิสต์โดยไม่ผ่านสมาชิกตัวที่ 1 ถึง $n-1$ ซึ่งแตกต่างกับอาร์เรย์ ที่สามารถอ้างถึงสมาชิกตัวใดๆ ในลิสต์นั้นได้เลย ดังตัวอย่าง



จากภาพมี $head$ เป็นพอยเตอร์ชี้ไปที่ต้น list

$head -> info$

คือ ข้อมูลที่อยู่ในโหนดที่

$head$ ชี้ คือ A

$head -> link$

คือ โหนดที่อยู่ถัดจากโหนดที่

ชี้ด้วย $head$ ชี้ คือ โหนดที่ 2

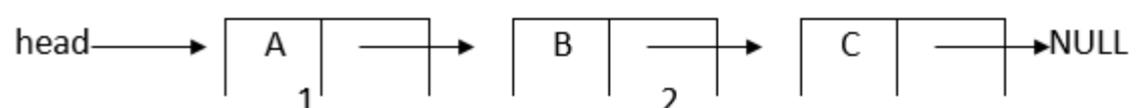
$head -> link -> info$ คือ B

$head -> link -> link$ คือ โหนดที่ 3

$head -> link -> link -> info$ คือ C

$head -> link -> link -> link$ คือ NULL

จะเห็นว่าเป็นการไม่สะดวกนัก ถ้าจะไปถึงข้อมูลโหนดสุดท้ายๆ โดยเฉพาะอย่างยิ่งถ้ามีข้อมูลมากๆ ดังนั้น ควรจะมีตัวชี้变量นอกที่คอยเลื่อนไปยังโหนดที่ต้องการจะเข้าถึงข้อมูลในโหนด ดังตัวอย่าง

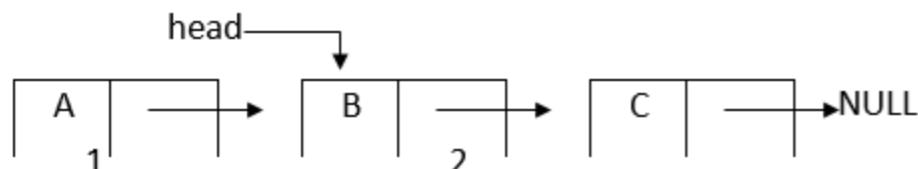


head -> info คือ A

head -> link คือ โหนดที่ 2

ถ้าจะเลื่อนไปชี้ยังโหนดที่ 2 สามารถทำได้โดยสั่งเลื่อนตัวชี้ภายนอกที่เป็นตัวชี้โหนดให้ชี้ไปที่โหนดถัดไป (โหนดที่ 2) โดยสั่ง

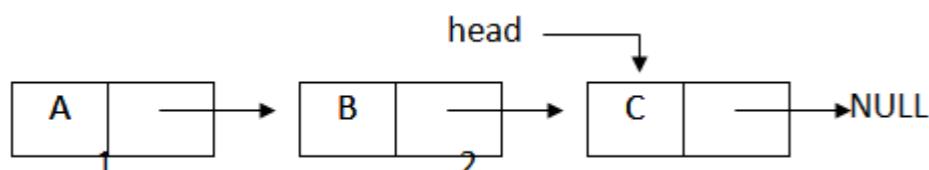
head = head -> link;



head -> info คือ B

head -> link คือ โหนดที่ 3

เช่นเดียวกัน ถ้าจะเลื่อน head ไปชี้ยังโหนดถัดไป ก็ต้องสั่ง head = head -> link;



head -> info คือ C

head -> link คือ NULL

จากตัวอย่างข้างต้น ถ้าต้องการท่องไปในลิสต์อีกรอบก็จะทำไม่ได้ เพราะไม่รู้ว่าต้นลิสต์อยู่ที่ใด ดังนั้นเราจึงต้องใช้ตัวชี้ภายนอกอีกตัวหนึ่งเป็นตัวที่ใช้ท่องเข้าไปในลิสต์ เพื่อให้ head ยังคงเก็บตำแหน่งของโหนดแรกเอาไว้

ถ้ากำหนดให้ current เป็นพอยเตอร์ที่กำหนดขึ้นมาเพื่อใช้ในการท่องไปในลิสต์ ดังนั้นในการเริ่มต้น current จึงต้องชี้ไปที่โหนดเช่นเดียวกับ head ซึ่งสามารถกำหนดได้ด้วยประโยค

current = head;

และถ้าจะเลื่อน current ไปยังโหนดถัดไปให้กำหนด

current = current -> link;

ตัวอย่างฟังก์ชันที่ใช้ในการแสดงข้อมูลในลิสต์ทั้งหมด

```
void DisplaySingly (void) {
```

```
    if ( head == NULL )
```

```
        printf ("EMPTY LIST");
```

```
    else {
```

```
        current = head;
```

```
        while ( current != NULL ) {
```

```

printf ("%d ", current -> info);

current = current -> link;

}

}

}

```

2. ลิงค์ลิสต์คู่ (Doubly Linked List) จากลิงค์ลิสต์ เดียวที่กล่าวมาแล้ว เป็นลิสต์ที่มีการเชื่อมต่อทางเดียว การติดต่อกับข้อมูลต่างๆ จะทำได้ตามลำดับทางเดียว โดยเริ่มต้นจากพอยเตอร์ head ซึ่งชี้ไปยังโหนดแรกของการ และลิงค์ของโหนดแรกชี้ไปยังโหนดถัดไป ซึ่งถ้าทำงานอยู่ที่โหนด N ในลิสต์ เราสามารถติดต่อกับโหนดที่อยู่ถัดไปได้โดยใช้พอยเตอร์ของโหนด N แต่เราไม่สามารถติดต่อกับโหนดที่อยู่ก่อนหน้าโหนด N ได้โดยตรง หากต้องการติดต่อกับโหนดที่อยู่ก่อนหน้าโหนด N จะต้องเริ่มจากโหนดแรกของลิสต์

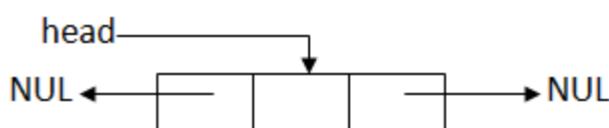
แต่สำหรับลิงค์ลิสต์คู่นี้ จะเป็นลิงค์ลิสต์ที่แต่ละโหนดมีการแสดงออกถึงลำดับก่อนหลังอย่างชัดแจ้ง โดยในแต่ละโหนดจะมี 2 ลิงค์ฟิลด์ ดังรูป

llink info rlink



1. ส่วนของ info ใช้สำหรับเก็บข้อมูลข่าวสารของโหนด
2. ส่วนของ llink ซึ่งเป็นพอยเตอร์ใช้สำหรับเก็บตำแหน่งของโหนดที่อยู่ก่อนหน้า ถ้าไม่ชี้ไปที่โหนดใดก็จะมีค่าเป็น NULL
3. ส่วนของ rlink ซึ่งเป็นพอยเตอร์ใช้สำหรับเก็บตำแหน่งของโหนดที่อยู่ถัดไป ถ้าไม่ชี้ไปที่โหนดใดก็จะมีค่าเป็น NULL

และเช่นเดียวกันกับลิงค์ลิสต์เดียวคือ จะต้องมีตัวชี้ภาพนอกราบเพื่อเก็บตำแหน่งของโหนดแรกของลิสต์



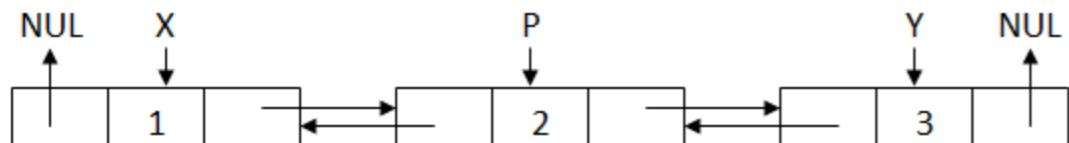
การเรียกส่วนประกอบต่างๆ ที่เกี่ยวข้องกับลิงค์ลิสต์คู่

head -> info หมายถึง ส่วนของข่าวสารของโหนดที่ถูกชี้โดยตัวชี้ head

head -> llink หมายถึง ส่วนของตำแหน่งของโหนดก่อนหน้า
โหนดที่ถูกซีดโดยตัวชี้ head ในที่นี่คือ NULL

head -> rlink หมายถึง ส่วนของตำแหน่งของโหนดถัดไปที่
ถูกซีดโดยตัวชี้ head ในที่นี่คือ NULL

ตัวอย่างเช่น ให้ X, P และ Y ชี้ไปยังแต่ละโหนด ดัง
รูป



X -> rlink คือ โหนด P

P -> rlink คือ โหนด Y

Y -> llink คือ โหนด P

P -> llink คือ โหนด X

ดังนั้น

P -> rlink -> llink = P -> llink -> rlink;

การสร้างลิงค์ลิสต์คู่

การประ公示 โครงสร้างของโหนดในลิงค์ลิสต์คู่ สามารถ
ทำได้ดังนี้

```
typedef struct list {
```

```
    struct list *llink;
```

```
    int      info;
```

```
    struct list *rlink;
```

```
}node;
```

```
node      *head, *temp, *current, *pred;
```

พิจารณาการประ公示 โครงสร้างของลิสต์ข้างต้น แสดงว่า
ในโหนดจะประกอบไปด้วย 3 พลัดย่อย คือ

พลัดตัวชี้ทางซ้าย (Linked Field) ได้แก่ llink

พลัดข้อมูล (Data Field) ได้แก่ inf

พลัดตัวชี้ทางขวา (Linked Field) ได้แก่ rlink

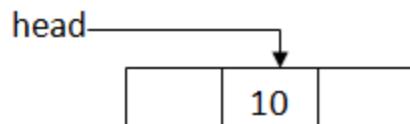
โดยพลัดตัวชี้ทางซ้ายจะทำหน้าที่เก็บตำแหน่งของโหนด
ที่อยู่ทางซ้าย และพลัดตัวชี้ทางขวาจะทำหน้าที่เก็บตำแหน่ง
ของโหนดที่อยู่ถัดไป เมื่อประ公示 โครงสร้างของโหนด

เรียบร้อยแล้ว ในขั้นตอนต่อไปคือการสร้าง โหนดแรกของลิสต์ ซึ่งมีกระบวนการดังนี้

ในกรณีที่ยังไม่มีโหนดเลย ตัวชี้ภายในออกทุกตัวจะเก็บค่า NULL เช่นเดียวกันกับลิงค์ลิสต์เดี่ยว ดังนี้

head → NULL

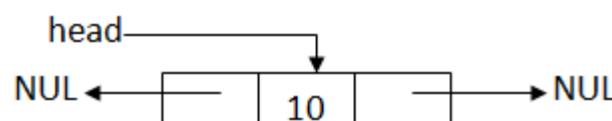
ยืม โหนดว่าง จาก Storage Pool โดย ให้พอยเตอร์ head ชี้ไปที่ โหนดว่างนั้นแล้ว ใส่ข้อมูล



head = NEW node

head -> info = 10;

ให้พอยเตอร์ทางซ้ายและทางขวาของ โหนดใหม่ชี้ไปที่ NULL



head -> llink = head -> rlink = NULL;

ตัวอย่างฟังก์ชันที่ใช้ในการสร้าง โหนดแรกของลิงค์ลิสต์เดี่ยว

```
void CreateDoubly (void) {  
    if ( head == NULL )  
        head = NEW node;  
    head -> info = X;  
    head -> llink = head -> rlink = NULL;  
}
```

การแสดงข้อมูล ในลิงค์ลิสต์คู่

การแสดงข้อมูล ในลิงค์ลิสต์คู่ สามารถทำได้ 2 แบบ คือ แสดงจากข้อมูลแรกไปหาข้อมูลสุดท้าย โดยการท่องลิสต์ จากต้นลิสต์ไปท้ายลิสต์ โดยใช้ rlink ของแต่ละโหนด และ แสดงข้อมูลสุดท้ายมาหาข้อมูลแรก โดยท่องจากท้ายลิสต์กลับ มาที่ต้นลิสต์ โดยใช้ llink ของแต่ละโหนดและถ้าลิงค์ลิสต์นั้น เป็นลิสต์ที่เรียงข้อมูลจากน้อยไปมาก การท่องลิสต์จากต้นลิสต์ไปท้ายลิสต์จะได้ข้อมูลที่เรียงจากน้อยไปมาก และ การท่องลิสต์จากท้ายลิสต์กลับมาต้นลิสต์ก็จะทำให้ได้ข้อมูลที่เรียงจากมากไปน้อย

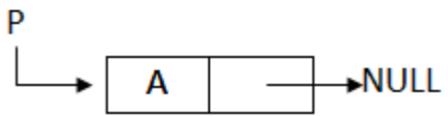
ตัวอย่างฟังก์ชันที่ใช้ในการพิมพ์ข้อมูลในลิงค์ลิสต์คู่จากซ้ายไปขวา

```
void DisplayDoubly (void) {  
    if ( head == NULL )  
        printf ("EMPTY LIST");  
    else {  
        current = head;  
        while ( current != NULL ) {  
            printf ("%d ", current -> info);  
            current = current -> rlink;  
        }  
    }  
}
```

ในการนี้ที่ไม่มีสมาชิกในลิสต์เลย ตัวแปร head จะต้องเก็บค่า NULL ไว้ ซึ่งรายการเชื่อมโยงแบบนี้ เรียกว่ารายการว่าง (null list) ดังภาพ

head → NULL

การเรียกส่วนประกอบต่างๆ ที่เกี่ยวข้องกับลิงค์ลิสต์เดียว



จากการพัฒนาให้ P เป็น ตัวชี้ (Pointer) ไปยังโนนด

P -> info คือ ข้อมูลที่อยู่ในโนนดที่ถูกชี้ด้วยตัวชี้ P ซึ่งก็คือ A

P -> link คือ ที่อยู่ของโนนดถัดจากโนนดที่ชี้ด้วยตัวชี้ P ซึ่งก็คือค่า NULL

ในการกำหนดค่าให้กับตัวชี้ไม่ว่าจะเป็นตัวชี้ภายในหรือตัวชี้ภายนอก ค่าที่จะนำมากำหนดให้ต้องเป็นประเภทเดียวกันคือ ต้องเป็นตำแหน่ง (address) หรือ ค่า NULL เท่านั้น เนื่องจากตัวชี้จะเก็บค่าที่เป็นตำแหน่งของโนนด ยกตัวอย่างเช่น ถ้าให้ P และ R เป็นตัวชี้ภายนอกที่ชี้ไปยังโนนด และ Q เป็นตัวชี้อีกด้วยหนึ่ง ถ้าต้องการให้ Q ชี้ไปที่โนนดเดียวกันกับที่ P ชี้อยู่ จะสามารถทำได้โดยตรง โดยการสั่งให้ Q = P เนื่องจากทั้ง P และ Q เป็นตัวชี้ทั้งคู่ ดังตัวอย่าง



เมื่อสั่ง $Q = P$ จะได้ผลดังนี้



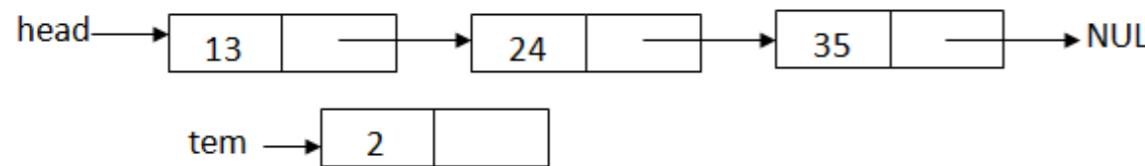
หรือถ้าต้องการให้โนนด **R** เชื่อมกับโนนด **P** โดยให้
อยู่ต่อจากโนนด **P** สามารถทำได้โดยกำหนดให้ link ของ
โนนด **P** ชี้ไปที่ตำแหน่งเดียวกับที่ตัวชี้ **R** ชี้อยู่ โดยเมื่อสั่ง
 $P \rightarrow \text{link} = R$ จะได้ผลดังนี้



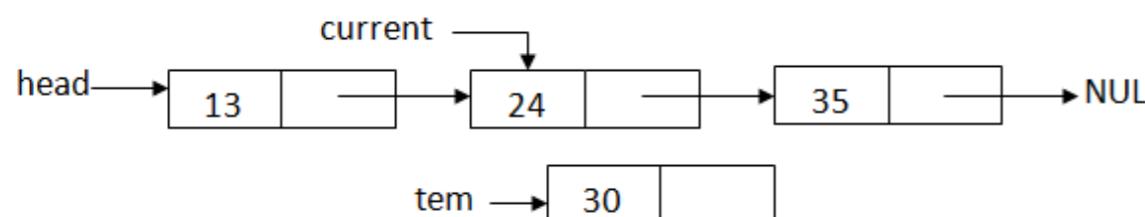
Section 3

แบบฝึกหัด

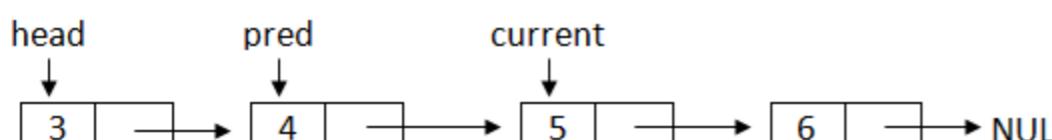
1. ต้องการเชื่อมโหนดใหม่ (โหนดที่ temp ชี้อยู่) ให้เป็นโหนดแรกของรายการ (List) ต้องส่งอย่างไรบ้าง



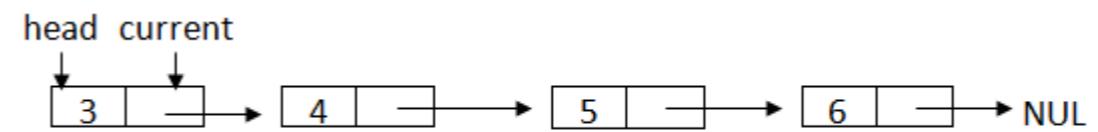
2. ต้องการเชื่อมโหนดใหม่ (โหนดที่ temp ชี้อยู่) ให้เป็นโหนดถัดจากโหนด current ต้องส่งอย่างไรบ้าง



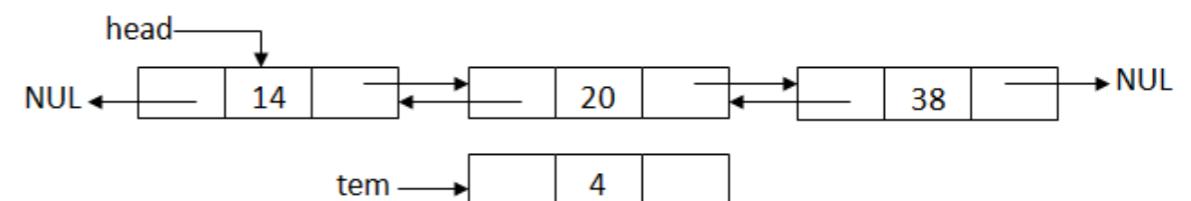
3. ต้องการลบโหนดที่ current ชี้อยู่ ต้องส่งอย่างไรบ้าง



4. ต้องการลบโหนดแรกของรายการ (List) ต้องส่งอย่างไรบ้าง



5. ต้องการเชื่อมโหนดใหม่ (โหนดที่ temp ชี้อยู่) ให้เป็นโหนดแรกของรายการ (List) ต้องส่งอย่างไรบ้าง



Stack Structure



วัตถุประสงค์

- 1.เข้าใจแนวคิดและหลักการทำงานของโครงสร้างข้อมูลแบบสเตก
- 2.อธิบายหลักการทำงานของฟังก์ชันดำเนินงานพื้นฐานของสแตกได้
- 3.เข้าใจหลักการสร้างสแตกด้วยอาร์เรย์และลิงค์สิลต์ได้
- 4.สามารถแปลงนิพจน์ Infix ให้เป็น Postfix ได้
- 5.เข้าใจหลักการทำงาน โปรแกรมรีเคอร์ชีฟ

นิยามของสแตก

โครงสร้างแบบอาร์เรย์และลิงค์ลิสต์ เป็นโครงสร้างที่เราสามารถแทรกหรือลบอิลิเม้นท์ในตำแหน่งใดๆ ของรายการได้ตามต้องการ แต่มีการใช้งานหลายอย่างที่เราต้องการเฉพาะการแทรกหรือการลบข้อมูลในตำแหน่งสุดท้ายเท่านั้น ซึ่งโครงสร้างข้อมูลที่ใช้ในงานลักษณะนี้ คือ โครงสร้างสแตก



สแตกเป็นโครงสร้างข้อมูลที่ถูกกล่าวถึงมาก โครงสร้างหนึ่ง ซึ่งมักจะใช้เป็นประโยชน์ในการอินเตอร์รัปต์ การกระโดดไปมาระหว่างโปรแกรมย่อย การเขียนโปรแกรมแบบเรียกใช้ตัวเอง (*recursive*) นอกจากนั้นแล้ว โครงสร้างข้อมูลชนิดนี้มักจะใช้ช่วยในการเข้าไปในโครงสร้างแบบพิเศษ เช่น เครือข่าย หรือต้นไม้ โดยจะช่วยในการจำเส้นทาง และงานที่เราำโครงสร้างแบบสแตกแล้วลบเพ็บบอยๆ คือ การยกเลิกคำสั่ง (*Undo*) ในไมโครซอฟท์ เวิร์ด นั่นเอง

โครงสร้างข้อมูลแบบสแตก

สแตกเป็นโครงสร้างแบบเชิงเส้น ที่มีลักษณะที่ว่า การนำข้อมูลเข้าสู่สแตก (insertion) และการนำข้อมูลออกจากสแตก (deletion) สามารถที่จะทำได้ที่ปลายด้านหนึ่งของลิสต์ที่แทนสแตกเท่านั้น ดังนั้นอันดับของการนำสมาชิกเข้าและออกจากสแตกมีความสำคัญ คือ สมาชิกที่เข้าไปอยู่ในสแตกก่อนจะออกจากสแตกหลังสมาชิกที่เข้าไปใน สแตกทีหลัง นั่นคือ การเข้าทีหลังออกก่อน จึงเรียกลักษณะแบบนี้ว่า LIFO (Last In First Out)



สแตกประกอบด้วยส่วนสำคัญ ๆ 2 ส่วน คือ

- 1. ตัวชี้สแตก หรือ Stack Pointer** ซึ่งเป็นตัวควบคุมการนำสมาชิกเข้า หรือออกจากสแตก เป็นตัวใช้บอกร่างสแตกนั้นเต็มหรือยัง
- 2. ส่วนสมาชิกของสแตก** หรือจะเรียกอีกอย่างว่า Stack Element สมาชิกของสแตกนี้จะเป็นข้อมูลชนิดเดียวกันทั้งหมด

การสร้างสแตก

ในการแทนโครงสร้างข้อมูลแบบสแตกจะใช้โครงสร้างข้อมูลแบบอาร์เรย์ หรือลิงค์ลิสต์ก็ได้ ทั้งนี้แล้วแต่ความเหมาะสม สมในการนำไปใช้ในการทำงาน ถ้าใช้การแทนที่ข้อมูลของสแตกด้วยอะเรย์ซึ่งเป็นการจัดสรรเนื้อที่หน่วยความจำแบบสแตกติก ก็จะต้องมีการกำหนดขนาดของสแตกล่วงหน้าว่าจะมีขนาดเท่าใด แต่ถ้าเลือกการแทนที่ข้อมูลของสแตกด้วยลิงค์ลิสต์ซึ่งเป็นการจัดสรรเนื้อที่หน่วยความจำแบบไดนามิก สแตกจะไม่มีวันเต็มตราบใดที่ยังมีเนื้อที่ในหน่วยความจำ นั่นคือ หน่วยความจำจะถูกจัดสรรเมื่อมีการขอใช้จริงๆ ระหว่างการ

ประมวลผลโปรแกรมผ่านตัวแปรชนิด pointer แต่ในที่นี้จะกำหนดให้ตัวสแตกเป็นแบบอาร์เรย์

นอกจากตัวสแตกเองแล้ว ในการทำงานกับสแตกยังต้องกำหนดตัวแปรเพิ่มอีกหนึ่งตัวเพื่อกีบตัวชีสแตก (Stack Pointer) โดยเริ่มแรกในขณะที่สแตกยังไม่มีข้อมูล ตัวชีสแตก ก็จะซีที่ 0 (ยังไม่ได้ซีที่ซ่องใดๆ ในสแตก)

การดำเนินงาน

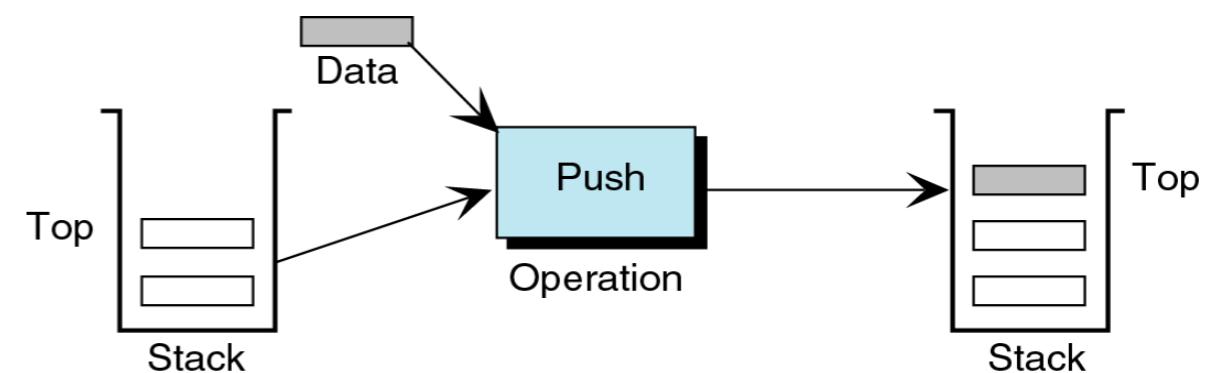
ทำงานกับโครงสร้างข้อมูลแบบสแตกได้แก่ การ PUSH และการ POP

การ PUSH

เป็นการกระทำหรือการทำงานของสแตกที่นำข้อมูลเข้าสู่สแตก โดยก่อนที่จะนำข้อมูลเข้ามานั้นจะต้องมีการจัดการให้ตัวชีสแตกซึ่ไปยังช่องหรือตำแหน่งต่อไปของส่วนของตัวสแตก ก่อน ซึ่งเป็นช่องหรือตำแหน่งที่ว่างอยู่ไม่มีข้อมูล แล้วจึงค่อยทำการ PUSH ข้อมูลลงสู่สแตกในตำแหน่งที่ตัวชีสแตกซื้อยู่

ในกรณีที่ PUSH ข้อมูลลงสู่สแตก จนตัวชีสแตกเท่ากับจำนวนช่องของสแตกแล้ว จะไม่สามารถทำการ PUSH ข้อมูลลงสแตกได้อีก เนื่องจากตัวชีสแตกไม่สามารถที่จะขยายไปยังช่องต่อไปได้ จะเกิด Error ที่เรียกว่า Stack Overflow

ตัวอย่าง เพิ่มข้อมูลใน Stack: Push



ALGORITHM PUSH

เพื่อใช้ในการแทรกข้อมูล x เข้า Stack โดยที่

TOP แทน ตัวชีสแตก

N แทน จำนวนช่องของสแตก

X แทน ข้อมูล

ST แทน สแตก

1. [ตรวจสอบ Overflow]

IF TOP >= N THEN

PRINT " STACK OVERFLOW "

EXIT

ENDIF

2. [เพิ่มค่า Stack Pointer]

TOP = TOP + 1

3. [แทรกข้อมูลเข้า Stack]

ST (TOP) = X

4. [จบการทำงาน]

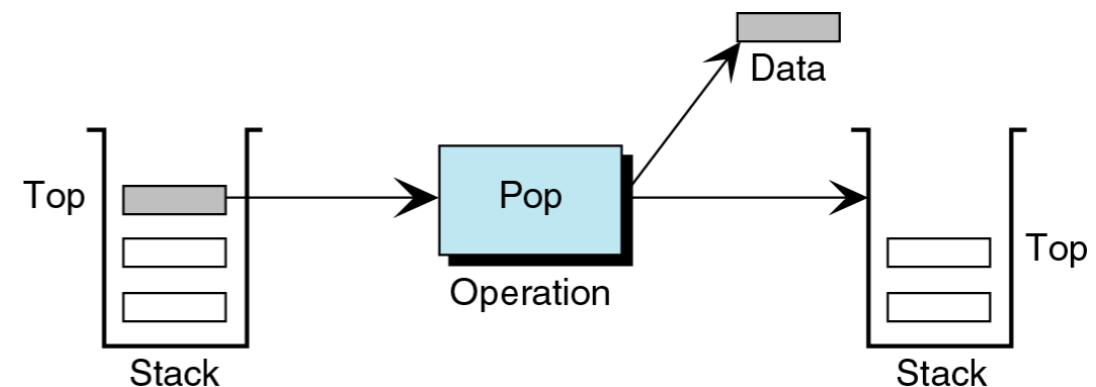
EXIT

การ POP

เป็นการกระทำหรือการทำงานของสแตกที่นำข้อมูลที่เก็บอยู่ในสแตกออกจากสแตกมาใช้งาน โดยการ POP นี้ เมื่อทำการ POP ข้อมูลนั้นออกจากสแตกแล้ว จะต้องมีการจัดการให้ตัวชี้สแตกซึ่งป้ายังช่องหรือตำแหน่งก่อนหน้าข้อมูลที่ได้ทำการ POP ข้อมูลออกไป

การ POP ข้อมูลนี้จะทำการนำข้อมูลในส่วนบนสุดของสแตกออกไปทำงานตามต้องการ แต่การ POP ข้อมูลนี้จะไม่สามารถ POP ข้อมูลออกจากสแตกที่ว่างเปล่าหรือไม่มีข้อมูลได้ ถ้าเราพยายาม POP ข้อมูลออกจากสแตกที่ว่างเปล่า จะเกิด Error ที่เรียกว่า Stack Underflow

ตัวอย่าง ลบข้อมูลในสแตก : Pop



ALGORITHM POP

เพื่อใช้ในการลบข้อมูล x จาก Stack โดยที่

TOP แทน ตัวชี้สแตก

N แทน จำนวนช่องของสแตก

Y แทน ข้อมูล

ST แทน สแตก

1. [ตรวจสอบ Underflow]

IF TOP <= 0 THEN

PRINT " STACK UNDERFLOW "

EXIT

ENDIF

2. [นำข้อมูลที่ต้องการออกจาก Stack เก็บไว้ที่ Y]

Y = ST(TOP)

3. [ลดค่า Stack Pointer]

TOP = TOP - 1

4. [จบการทำงาน]

EXIT

เปรียบเทียบประสิทธิภาพของการสร้างสแตกด้วยอะเรย์ และลิงค์ลิสต์

การเลือกการแทนที่ข้อมูลสแตกด้วยอะเรย์ มีข้อจำกัด สำหรับขนาดของสแตกและจะต้องจองเนื้อที่เท่ากับขนาดที่ใหญ่ที่สุดของสแตกไว้เลย เพราะเป็นการจัดสรรเนื้อที่ในหน่วยความจำแบบสแตกติก

ส่วนการเลือกการแทนที่ข้อมูลสแตกด้วยลิงค์ลิสต์ ไม่มีข้อจำกัดของขนาดของสแตกและหน่วยความจำจะถูกใช้ก็ต่อเมื่อมีข้อมูลจริงๆ และว่างหน้า เพราะเป็นการจัดสรรเนื้อที่หน่วยความจำแบบไดนามิก ซึ่งทำให้ประหยัดเนื้อที่ในหน่วยความจำมากกว่า แต่การเขียนโค้ดสำหรับการแทนที่ข้อมูลสแตกด้วยอะเรย์ง่ายกว่าการแทนที่ข้อมูลด้วยลิงค์ลิสต์

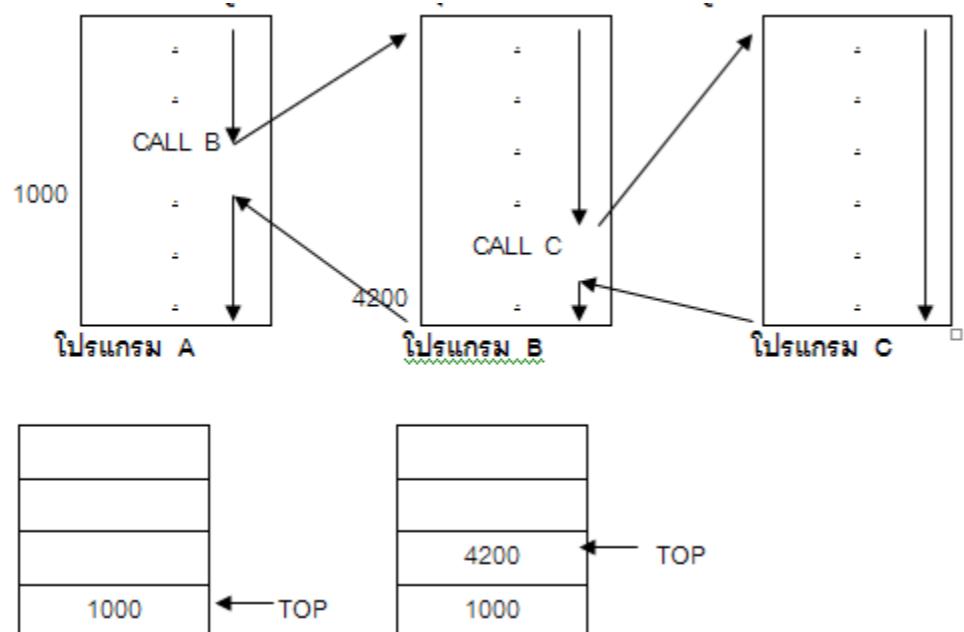
Section 2

การประยุกต์ใช้งานสแตก

1. การจัดการหน่วยความจำ

2. การใช้สแตกในกระบวนการเรียกใช้โพรชีเจอร์หรือฟังก์ชัน

สแตกเป็นโครงสร้างข้อมูลที่มีประโยชน์มาก ถูกนำไปใช้ทั้งในซอฟต์แวร์ระบบ (System Software) และในการประยุกต์โดยยูสเซอร์ (user) เช่น ช่วยคอมไพล์เวอร์ (Compiler) สร้างกฎเกณฑ์ของโปรแกรมมิ่งเกี่ยวกับการเรียกโปรแกรมย่อย (Subprogram call) ที่ว่า โปรแกรมย่อยได้ที่ถูกเรียกทำงานที่หลังสุด ต้องทำงานเสร็จก่อน ดังภาพ



ภาพแสดงลักษณะการทำงานในสแตกในการเรียกใช้ฟังก์ชัน

จากรูปแสดงการเรียกใช้โปรแกรมย่อย B และ C โดยในโปรแกรมหลัก A มีคำสั่งเรียกโปรแกรมย่อย B และในโปรแกรมย่อย B มีคำสั่งเรียกโปรแกรมย่อย C โปรแกรมย่อย C ต้องถูกกระทำเสร็จก่อน ตามมาด้วยโปรแกรมย่อย B และโปรแกรมหลัก A ซึ่งลำดับของการทำงานของคำสั่งแสดงด้วยลูกศร โดยสแตกช่วยเก็บที่อยู่ของคำสั่งถัดจากคำสั่งเรียกใช้โปรแกรมย่อย ซึ่งคำสั่งนี้จะเป็นคำสั่งที่ถูกทำงานต่อหลังจากได้ทำงานตามคำสั่งในโปรแกรมย่อยที่เรียกไป จากรูปสมมติว่า ในขณะทำงานคำสั่งถัดจาก CALL B ในโปรแกรมหลัก A อยู่ ณ แอดเดรส 1000 และที่อยู่ของคำสั่งถัดจาก CALL C ในโปรแกรมย่อย B อยู่ ณ แอดเดรส 4200 เมื่อโปรแกรมหลัก A ทำงานมาถึงคำสั่ง CALL B แอดเดรส 1000 จะถูก PUSH ลงสู่สแตก และเช่นกันเมื่อโปรแกรมย่อย B ถูกทำงานมาถึงคำสั่ง CALL C แอดเดรส 4200 จะถูก PUSH ลงสู่สแตก ดังรูป ดังนั้นหลังจากการทำงานตามคำสั่งในโปรแกรมย่อย C จนหมดแล้ว แอดเดรสของคำสั่งถัดไปที่จะถูกทำงานจะถูก POP ออกจากสแตก คือคำสั่งที่แอดเดรส 4200 และเมื่อจบโปรแกรมย่อย B คำสั่งที่จะถูก

ทำงานต่อไปจะถูก POP ออกจากสแตกเช่นเดียวกัน คือคำสั่งที่แสดงผล 1000

IF(1).....

IF(2).....

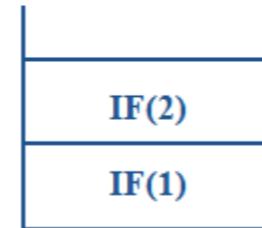
IF(3).....

.....

ELSE(3)....

ELSE(2).....

ELSE(1).....



ภาพแสดงการประยุกต์ใช้สแตกในฟังก์ชันเกี่ยวกับ IF-EISE

3. การตรวจสอบอักขระสมดุล (Balancing Symbol)

ผู้ที่มีประสบการณ์ในการเขียนโปรแกรมมาแล้ว จะพบว่า สิ่งที่เรามักจะหลงลืมเมื่อเขียนโปรแกรม และทำให้เกิดข้อผิดพลาดอยู่บ่อย ๆ คือ การหลงลืมอักขระสมดุล เช่น { คู่กับ }, [คู่กับ], (คู่กับ) เป็นต้น ซึ่งในการตรวจสอบอักขระสมดุลนั้น คอมไพลเลอร์นำชนิดข้อมูลแบบสแตกมาประยุกต์ใช้ได้ โดยมีวิธีการดังนี้ให้อ่านอักขระทีละตัว

ถ้า อักขระเป็นอักขระเปิด เช่น {, [, เป็นต้น ให้ Push ลงสแตก

ถ้า อักขระเป็นอักขระปิด เช่น },],), เป็นต้น ให้ตรวจ สอบว่าอักขระบน Top ของสแตกเป็นอักขระเปิดที่คู่กันหรือไม่

ถ้าใช่ ให้ Pop ออกจนน้อยกว่าจากสแตก แต่ถ้าไม่ใช่ แสดงผล error

เมื่ออ่านอักขระหมดแล้ว แต่สแตกไม่เป็นสแตกわ่าง ให้แสดงผล error

?
((A + B) / C

(a) Opening parenthesis not matched

?
(A + B) / C)

(b) Closing parenthesis not matched

ภาพแสดงการประยุกต์ใช้สแตกในการตรวจสอบอักขระสมดุล

4. การแปลงนิพจน์ infix ให้เป็น postfix

โดยปกติเวลาเขียนโปรแกรมสั่ง ให้เครื่องคำนวณ ต้องเขียนนิพจน์ที่ต้องการไว้ในตัวโปรแกรม ซึ่งนิพจน์เหล่านี้ เรียกว่า นิพจน์ infix คือนิพจน์ที่มี ตัวดำเนินการ (Operator) อยู่ระหว่างตัวถูกกระทำ (Operand) เช่น A + B เครื่องหมาย + เป็นโอเปอเรเตอร์ระหว่างโอเปอර์แรนด์ A และ B ซึ่งเห็นว่าเป็นนิพจน์ที่มีนุษย์คุ้นเคย

ตัวดำเนินการ ก็คือ เครื่องหมายทางคณิตศาสตร์ สำหรับการคำนวณต่างๆ เรียงตามลำดับการดำเนินการก่อน-หลัง (precedence) ได้แก่

ยกกำลัง \wedge

คูณ หาร $*$, $/$

บวก ลบ $+$, $-$

ถ้าเครื่องหมายมีลำดับการดำเนินการเดียวกัน จะเลือกดำเนินงานของเครื่องหมายจากซ้ายไปขวา (ยกเว้น ยกกำลัง) และถ้ามีวงเล็บจะดำเนินงานสิ่งที่อยู่ในวงเล็บก่อน

ข้อเสียของนิพจน์ infix ที่ทำให้คอมไพล์ยุ่งยาก คือ ลำดับความสำคัญของ ออเปอร์เรเตอร์(Precedence) ที่ต่างกัน เช่น เครื่องหมายยกกำลัง (\wedge) มีความสำคัญมากกว่า เครื่องหมายคูณ ($*$) และหาร ($/$) และเครื่องหมายคูณและหารมีความสำคัญมากกว่าเครื่องหมายบวก (+) และลบ (-) เครื่องหมาย ได้มีลำดับความสำคัญมากกว่าจะถูกคำนวณก่อน (ถ้าไม่มีวงเล็บกำกับ) เช่น $A + B * C$ เครื่องจะคำนวณ $B * C$ ก่อนแล้วนำผลลัพธ์นั้นไปบวกกับค่า A ซึ่งจะทำงานเหมือนกับ $A + (B * C)$ ส่วนนิพจน์ใดที่มีออเปอร์เรเตอร์ที่มีลำดับความสำคัญเท่ากัน การคำนวณจะกระทำการจากซ้ายไปขวา เช่น $A - B + C$ จะทำ $A - B$ ก่อน แล้วจึงนำผลลัพธ์นั้นไปบวกกับค่า C

เมื่อการประมวลผลนิพจน์ infix เป็นไปด้วยความยากที่ การคำนวณไม่เป็นไปตามลำดับของเครื่องหมาย ออเปอร์เรเตอร์ที่มีก่อนหลัง คอมไพล์เริ่งแบ่งนิพจน์ infix ให้เป็น นิพจน์ postfix เสียก่อน ซึ่งนิพจน์ postfix คือนิพจน์ที่มีออเปอร์เรเตอร์อยู่หลัง ออเปอร์แรนด์ทั้งสองของมัน เช่น

$AB +$	หมายถึง	$A + B$
$AB -$	หมายถึง	$A - B$
$AB ^$	หมายถึง	$A ^ B$
$AB *$	หมายถึง	$A * B$
$AB /$	หมายถึง	A / B

ข้อดีของนิพจน์ postfix คือเป็นนิพจน์ที่มีการคำนวณตาม ออเปอร์เรเตอร์ที่มากก่อนหลัง เช่น นิพจน์ $ABC^* +$ หมายถึง ทำการคูณแล้วจึงทำการบวก ซึ่งคือต้องคำนวณ B^*C ก่อน แล้วจึงนำผลลัพธ์นั้นไปบวกกับ A ต่อไป

Section 3

อัลกอริทึมแปลงนิพจน์ INFIX ให้เป็นนิพจน์ POSTFIX

เนื่องจากนิพจน์ infix มีลำดับความสำคัญของเครื่องหมายโอเปอร์เรเตอร์ซึ่งหมายความว่า โอเปอร์เรเตอร์ที่มาก่อน อาจจะไม่ใช่โอเปอร์เรเตอร์ที่ถูกประมวลผลก่อน ดังนั้น สแตกซึ่งมีคุณสมบัติเป็นไล่โพลิสต์จึงมีส่วนช่วยในการแปลงนิพจน์ infix ให้เป็นนิพจน์ postfix ในการนี้มีสิ่งที่เกี่ยวข้อง 3 อย่าง คือ

1. ข้อมูลเข้าซึ่งเป็นนิพจน์ infix
2. ข้อมูลออกหรือนิพจน์ postfix
3. สแตกที่ใช้เก็บโอเปอร์เรเตอร์

ข้อมูลเข้าจะถูกอ่านมาทีละอักขระ (character) แล้วดำเนินการต่อไปดังนี้

1. ถ้าข้อมูลเข้า (input character) เป็นโอเปอร์เรเตอร์ ให้พิมพ์ออกเป็นผลลัพธ์ (postfix string)
2. ถ้าข้อมูลเข้าเป็นโอเปอร์เรเตอร์ ให้ทำดังนี้

2.1 ถ้าสแตกยังว่างอยู่ ให้ PUSH โอเปอร์เรเตอร์ลงสแตก

2.2 ถ้าสแตกยังไม่ว่าง ให้เปรียบเทียบ โอเปอร์เรเตอร์ที่เข้ามากับ โอเปอร์เรเตอร์ที่ ท้อปของสแตก

2.2.1 ถ้า โอเปอร์เรเตอร์ที่เข้ามามี precedence น้อยกว่าหรือเท่ากับ โอเปอร์เรเตอร์ที่ท้อปของสแตก ให้ POP โอเปอร์เรเตอร์จากสแตกไปเป็นผลลัพธ์ และเปรียบเทียบกับ โอเปอร์เรเตอร์ที่ท้อปของสแตกต่อไป หยุดเมื่อ โอเปอร์เรเตอร์ที่เป็นข้อมูลเข้ามี precedence มากกว่า โอเปอร์เรเตอร์ที่ ท้อปของสแตก หลังจากนั้น ให้ PUSH ข้อมูลลงสแตก

2.2.2 ถ้า โอเปอร์เรเตอร์ที่เข้ามามี precedence มากกว่า โอเปอร์เรเตอร์ที่ท้อปของสแตก ให้ PUSH ลงสแตก

3. ถ้าข้อมูลเข้าเป็นวงเล็บเปิด ให้ PUSH ลงสแตก
4. ถ้าข้อมูลเข้าเป็นวงเล็บปิด ให้ POP ออกจากสแตกจนกว่าจะถึงวงเล็บเปิด และนำผลที่ POP ออกไปเป็นผลลัพธ์ โดยทิ้งวงเล็บปิดและเปิดทิ้งไว้

ถ้าข้อมูลหมด ให้ POP Operator ที่ยังคงเหลือในสแตก
ไปไว้เป็นผลลัพธ์จนสแตกว่าง

ตัวอย่างของการแปลงนิพจน์ Infix เป็น Postfix

1. นิพจน์ Infix : $A - B * C$

INPUT	STACK	OUTPUT
A	-	A
-	-	A
B	-	AB
*	-*	AB
C	-*	ABC
		ABC*-

นิพจน์ Postfix ที่ได้ คือ : $ABC*-$

2. นิพจน์ Infix : $A^* (B + C)$

INPUT	STACK	OUTPUT
A		A
*	*	A
(* (A
B	* (AB
+	* (+	AB
C	* (+	ABC
)	*	ABC+
		ABC+*

นิพจน์ Postfix ที่ได้ คือ : $ABC+*$

3.นิพจน์ Infix : A ^B ^ (C + D)

INPUT	STACK	OUTPUT
A		A
^	^	A
B	^	AB
^	^ ^	AB
(^ ^ (AB
C	^ ^ (ABC
+	^ ^ (+	ABC
D	^ ^ (+	ABCD
)	^ ^	ABCD +
		ABCD + ^ ^

นิพจน์ Postfix ที่ได้ คือ : ABCD+^^

นอกจากวิธีการแปลงนิพจน์ Infix เป็น Postfix ตามขั้นตอนแล้ว ยังสามารถทำได้ด้วยตนเองโดยไม่อาศัย Stack ก็ได้ ซึ่งมีวิธีการดังนี้

เข้าใจนิพจน์ Infix ให้ครบตามลำดับการคำนวณ
ข้อความที่ต้องคำนวณจะมีโครงสร้างเป็นแบบนี้
ตัวแปร ชี้ว่าต้องคำนวณตัวไหนก่อน ตัวไหนหลัง
ตัวแปรที่มีค่าคงที่ เช่น จำนวน เวลา สถานที่
เครื่องหมายดำเนินการ เช่น 加減乗除 ฯลฯ
เครื่องหมายวงเล็บ เปิด ให้หมด ก็จะได้นิพจน์ Postfix

การหาค่าผลลัพธ์จากนิพจน์ Postfix

การหาค่าทางคณิตศาสตร์จากนิพจน์ Postfix มีขั้นตอน
ให้ลองดู 2 ขั้นตอน คือ

- ถ้าเป็น Operand ให้ PUSH ลงสู่สแตก
- ถ้าเป็น Operator ให้ POP ค่า 2 ค่า จากสแตก และ^{*}
ดำเนินการโดยใช้ Operator ตัวนั้น ในการนี้ให้ใช้ค่าแรก
ที่ได้จากสแตกเป็น Operand ตัวที่ 2 จากนั้นก็เก็บค่า
ผลลัพธ์ในสแตก

ตัวอย่างของการแปลงหาค่านิพจน์ Postfix

นิพจน์ Postfix : 1 6 3 / 4 * + 7 -

INPUT	CALCULATE	STACK
1		1
6		1 6
3		1 6 3
/	6 / 3	1 2
4		1 2 4
*	2 * 4	1 8
+	1 + 8	9
7		9 7
-	9 - 7	2

ผลลัพธ์ที่ได้ คือ : 2

สรุปขั้นตอนในการหาผลลัพธ์ของนิพจน์ Postfix

ค้นหาเครื่องหมายดำเนินการทางซ้ายสุด ของนิพจน์ เลือกตัวถูกดำเนินการ 2 ตัว ที่อยู่ติดกับเครื่องหมายดำเนินการทางซ้าย ประมวลผลตามเครื่องหมายดำเนินการนั้นแทนที่ เครื่องหมายดำเนินการและตัวถูกดำเนินการ ด้วยผลลัพธ์ที่ได้

Section 4

รีเคอร์ชีฟ โปรแกรมมิ่ง (Recursive)



สแตกนอกจากจะใช้จัดการกับการเรียกใช้โปรแกรมย่อยแล้ว ในบางภาษาอย่างใช้จัดการการรีเคอร์ช (recursion) หรือการเรียกใช้ตัวเองในการเขียนโปรแกรมที่ต้องทำซ้ำซ้อน สามารถทำได้โดยการใช้หลักการทำซ้ำซ้อนด้วย LOOP การเขียนโปรแกรมแบบนี้เรียกว่า **โปรแกรมวนซ้ำ (iterative)** และแบบรีเคอร์ชีฟ (recursive) คือกระบวนการที่ฟังก์ชันหรือ **โปรแกรมเรียกตัวเองซ้ำๆ** จนกว่าจะถึงเงื่อนไขที่กำหนด ซึ่งจะใช้การประมวลผลแบบนี้กับการคำนวณ ที่แต่ละขั้นตอนอยู่ในรูปของผลลัพธ์ที่ได้จากขั้นตอนก่อนหน้า ปัญหาที่ต้องทำซ้ำ ส่วนมากจะเขียนในรูปแบบนี้ได้ เช่น การบังคับการโปรแกรมแบบรีเคอร์ชีฟคือ หลักการที่เรียกว่า รีเคอร์ชัน (recursion) ซึ่งหมายถึงการนิยามปัญหาหรือนิยามสูตรคณิตศาสตร์ของ

ฟังก์ชันแฟกทอเรียล

ผลคูณของเลขจำนวนเต็ม 1 ถึง n เรียกว่า n แฟกทอเรียล สามารถแสดงโดย $n!$

$$n! = 1 * 2 * 3 \dots (n-2) * (n-1) * n$$

เพื่อความสะดวก กำหนด $0! = 1$ ดังนั้น ฟังก์ชันจะไม่มีค่าเป็นลบ ซึ่งเราจะได้

$$0! = 1$$

$$1! = 1$$

$$2! = 1 * 2 = 2$$

$$3! = 1 * 2 * 3 = 6$$

$$4! = 1 * 2 * 3 * 4 = 24$$

$$5! = 1 * 2 * 3 * 4 * 5 = 120 \quad 6! = 1 * 2 * 3 * 4 * 5 * 6 = 720 \quad \text{และต่อๆไป}$$

สังเกต

$$5! = 5 * 4! = 5 * 24 = 120$$

และ

$$6! = 6 * 5! = 6 * 120 = 720$$

นั่นคือ ทุก ๆ ค่าของ n ที่เป็นบวกจะได้

$$n! = n * (n - 1)!$$

ดังนั้นฟังก์ชันแฟกทอเรียลจะกำหนดได้ดังนี้

(a) if $n = 0$ then $n! = 1$

(b) if $n > 0$ then $n! = n(n - 1)!$

สังเกตว่า尼ยามนี้ $n!$ เป็นรีקורסีฟ เนื่องจากมีการเรียกใช้ตัวเอง เมื่อใช้ $(n - 1)!$ แต่อย่างไรก็ตาม

ตัวอย่าง จงหาค่า $4!$

ในการนี้ที่ใช้วิธีการแก้ปัญหาแบบการวนซ้ำ ก็คือ

$$4! = 4 * 3 * 2 * 1 = 24$$

ในการนี้ที่ใช้วิธีการแก้ปัญหาแบบรีקורסีฟ ก็คือ

1. $4! = 4 * 3!$

2. $3! = 3 * 2!$

3. $2! = 2 * 1!$

4. $1! = 1 * 0!$

5. $0! = 1$

6. $1! = 1 * 1 = 1$

$$7. \quad 2! = 2 * 1 = 2$$

$$8. \quad 3! = 3 * 2 = 6$$

$$9. \quad 4! = 4 * 6 = 24$$

Step 1 การหาค่าของ $4!$ ทำได้โดยนำ $4 * 3!$ ซึ่งเรา
จะยังไม่หาค่าของ $4!$ จนกว่าจะทราบค่าของ $3!$

Step 2 การหาค่าของ $3!$ ทำได้โดยนำ $3 * 2!$ ซึ่งเรา
จะยังไม่หาค่าของ $3!$ จนกว่าจะทราบค่าของ $2!$

Step 3 การหาค่าของ $2!$ ทำได้โดยนำ $2 * 1!$ ซึ่งเรา
จะยังไม่หาค่าของ $2!$ จนกว่าจะทราบค่าของ $1!$

Step 4 การหาค่าของ $1!$ ทำได้โดยนำ $1 * 0!$ ซึ่งเรา
จะยังไม่หาค่าของ $1!$ จนกว่าจะทราบค่าของ $0!$

Step 5 กำหนดค่าของ $0!$ เท่ากับ 1

Step 6 เป็นการทำกลับโดยนำค่า $0!$ ซึ่งเท่ากับ 1 ไป
แทนในการหาค่า $1!$ จะได้ค่า $1! = 1 * 1 = 1$

Step 7 เป็นการทำกลับโดยนำค่า $1!$ ซึ่งเท่ากับ 1 ไป
แทนในการหาค่า $2!$ จะได้ค่า $2! = 2 * 1 = 2$

Step 8 เป็นการทำกลับโดยนำค่า $2!$ ซึ่งเท่ากับ 2 ไป
แทนในการหาค่า $3!$ จะได้ค่า $3! = 3 * 2 = 6$

Step 9 เป็นการทำกลับโดยนำค่า $3!$ ซึ่งเท่ากับ 6 ไป
แทนในการหาค่า $4!$ จะได้ค่า $4! = 4 * 6 = 24$

ถ้าให้ FACT (n) แทนฟังก์ชันที่ใช้คำนวณหาแฟกทอเรียลของ n จะเขียนได้ว่า

$$\text{FACT}(n) = n * \text{FACT}(n - 1)$$

ถ้าต้องการหา FACT (4) จะต้องผ่านการคำนวณ ดังนี้

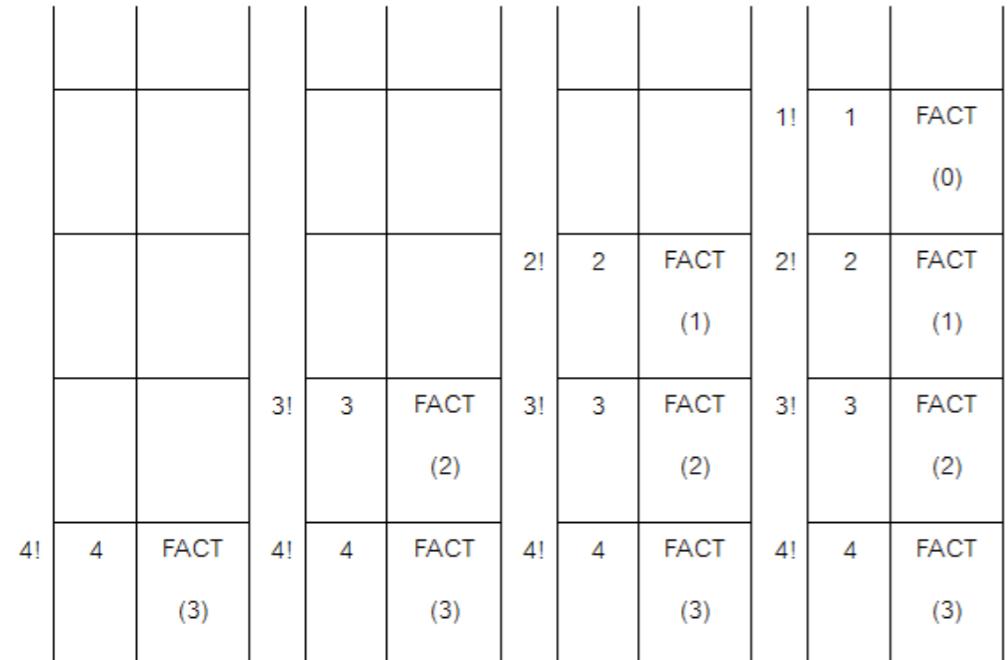
$$\text{FACT}(4) = 4 * \text{FACT}(3)$$

$$\text{FACT}(3) = 3 * \text{FACT}(2)$$

$$\text{FACT}(2) = 2 * \text{FACT}(1)$$

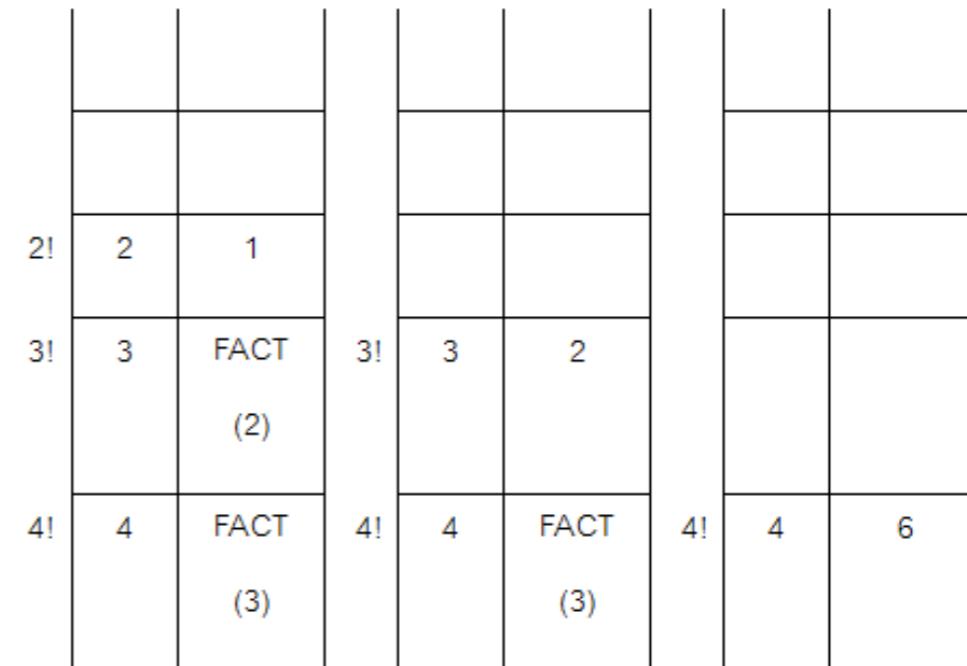
$$\text{FACT}(1) = 1 * \text{FACT}(0)$$

ภาพข้างล่างแสดงสแตกช่วยในการหา $n!$ เมื่อ $n = 4$ โดยที่สแตกเก็บค่า n และ $\text{FACT}(n-1)$ ไว้ในแต่ละครั้งที่มีการหาค่า $\text{FACT}(n)$ ได้ ๆ เนื่องจากในแต่ละขั้นยังไม่สามารถคำนวณได้ จนกว่าจะได้ค่า $\text{FACT}(0)$ จึงต้อง PUSH ค่า n และ $\text{FACT}(n - 1)$ ไว้ในสแตก เมื่อถึงจุดที่หาค่า $\text{FACT}(0)$ ซึ่งมีค่าเป็น 1



ภาพแสดงการใช้สแตกช่วยในการค้นหาข้อมูล

เมื่อหาค่า FACT (0) ได้แล้ว ก็ให้ POP ข้อมูลที่จะนำไปใช้ออกจากสแตก



ภาพแสดงการ Pop ข้อมูลที่ต้องการหาออกจากสแตก เพื่อนำข้อมูลไปใช้

แบบฝึกหัด

1. การสร้างสแตกด้วยอาร์เรย์มีข้อดี และข้อเสียอย่างไรบ้าง
2. จงบอกข้อจำกัดของ โครงสร้างข้อมูลแบบสแตก
3. จงแปลงนิพจน์ infix เป็น postfix ดังต่อไปนี้

❖ $A - B * C$

❖ $A * (B + C)$

❖ $A+B^*C-D$

❖ $(A+B)^*C-D$

❖ $A+(B-(C+D))^*E^2$

❖ $(A - 5 * (B + C) + D * E) * F$

4. หาตัวอย่างการใช้งานรีเครอร์ชีฟ พร้อมตัวอย่างการเขียนโปรแกรมแบบรีเครอร์ชีฟมา 2 ตัวอย่าง

5. ถ้ากำหนดให้ A , B , C และ D มีค่าเท่ากับ 2 , 3 , 4 และ 5 ตามลำดับ จงคำนวณการหาผลลัพธ์จากนิพจน์ Postfix ดังต่อไปนี้

d. $A B * C - D +$

e. $A B C + * D - S$

Queue Structure

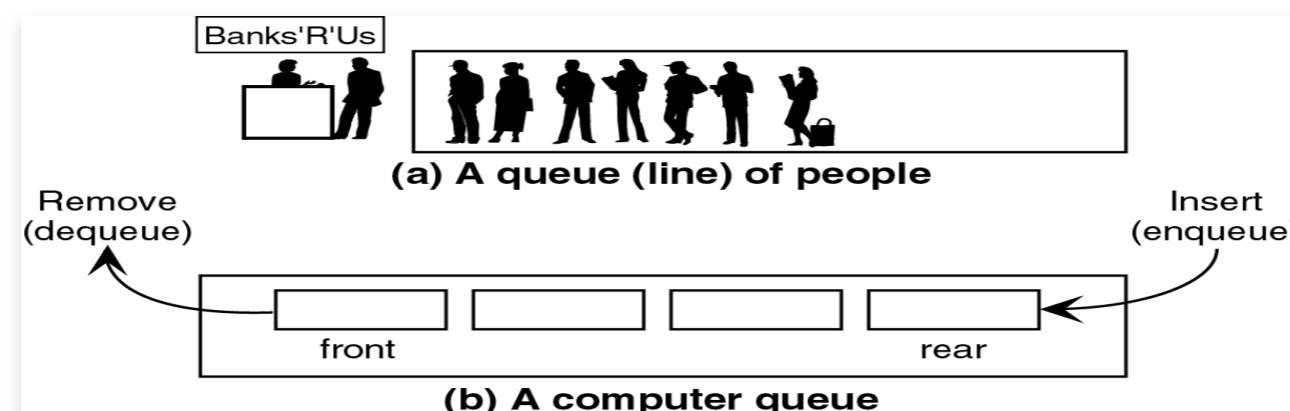


วัตถุประสงค์

- 1.เข้าใจแนวคิดและหลักการทำงานของโครงสร้างข้อมูลแบบคิว
- 2.สามารถยกตัวอย่างการประยุกต์โครงสร้างคิวที่นำมาใช้งานกับคอมพิวเตอร์ได้
- 3.บอกฟังก์ชันการดำเนินงานพื้นฐานของคิว
- 4.บอกเทคนิคการทำงานคิววงกลมได้

นิยามคิว

คิวเป็นโครงสร้างข้อมูลแบบหนึ่งซึ่งมีลักษณะที่ว่า ข้อมูลที่นำเข้าไปเก็บก่อนจะถูกนำออกมาทำงานก่อนส่วนข้อมูลที่เข้าไปเก็บทีหลังก็จะถูกนำออกมาใช้งานทีหลัง ขึ้นอยู่กับลำดับการเก็บข้อมูล จะเรียกลักษณะการทำงานแบบนี้ว่า เข้าก่อนออกก่อน หรือ *First In First Out (FIFO)* นั่นคือ โหนดที่เพิ่มเข้ามาในลิสต์ก่อนจะเป็นโหนดแรกที่ถูกดึงออกไป



การแทนที่โครงสร้างข้อมูลแบบคิว

การแทนที่โครงสร้างข้อมูลแบบคิว เราสามารถแทนที่ได้ 2 แบบเช่นเดียวกับ สเตก คือ การแทนที่คิวแบบสแตติก และ การแทนที่คิวแบบไดนามิก ดังนี้

1. การแทนที่คิวแบบสแตติก

การแทนที่คิวแบบสแตติกจะใช้แฉลามассив (Array) สำหรับสร้างคิว การแทนที่คิวด้วยวิธีนี้ต้องกำหนดขนาดคิวขึ้นมาก่อน และต้องใช้เนื้อที่ไม่เกินขนาดที่กำหนด โดยมีวิธีการประกาศโครงสร้างของคิวดังนี้

การดำเนินการกับคิวสามารถดำเนินการได้ใน 2 ลักษณะ ใหญ่ๆ คือ

1) การนำสมาชิกใหม่เข้าไปในคิว (insert) โดยจะแบ่งออกเป็น 2 กรณี คือ

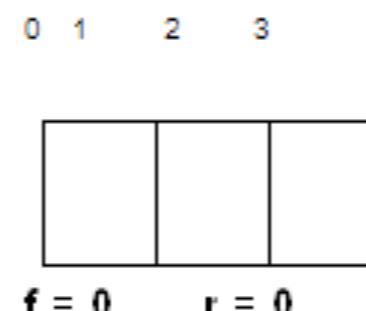
การนำข้อมูลเข้าไปในคิวว่าง โดยจะต้องดำเนินการให้พอยน์เตอร์ทั้ง 2 คือ ตัวชี้หัวคิวและตัวชี้หางคิว ซึ่งป้ายช่องแรกหรือตำแหน่งที่จะเก็บข้อมูลแรก

การนำข้อมูลเข้าไปในคิวต่อจากข้อมูลเดิม จะต้องจัดการให้ตัวชี้หางคิว ซึ่งป้ายช่องหรือตำแหน่งของข้อมูลที่นำเข้าไปล่วงตัวชี้หัวคิวยังคงซึ่งป้ายช่องหรือตำแหน่งของข้อมูลที่นำเข้าไปเป็นข้อมูลแรก

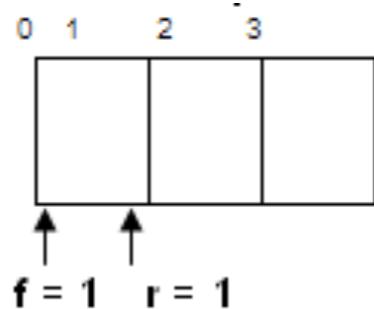
จากการ insert ข้อมูลเข้าไปในคิวแล้ว ถ้าหากทำการ insert ข้อมูลจนตัวชี้หางคิวยู่ที่ซองสุดท้ายแล้ว จะไม่สามารถทำการ insert ข้อมูลลงคิวได้อีก เนื่องจากตัวที่เก็บข้อมูลคิวเต็มทำให้ไม่สามารถที่จะรับข้อมูลอีก จึงได้มีฉันนั้นจะเกิดข้อผิดพลาด ที่เรียกว่า overflow ขึ้น เช่น ถ้าจองแฉลามассивขนาด 3 ซองซึ่งชื่อ q โดยมีตัวชี้หัวคิวคือ f และตัวชี้หางคิวคือ r

ตัวอย่าง

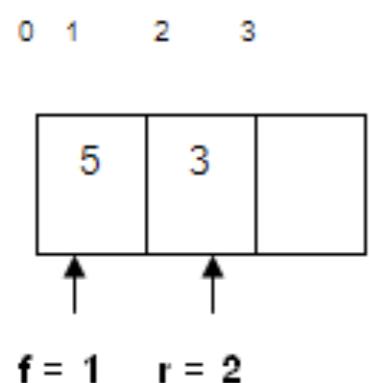
1. เมื่อคิวว่าง (empty queue)



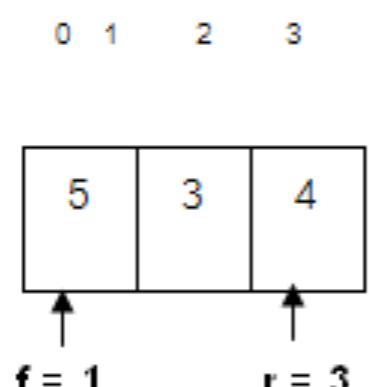
2. insert (5)



3. insert (3)



4. insert (4)



หลังจากนี้จะเพิ่มข้อมูลเข้าคิวอีกไม่ได้ เนื่องจากคิวเต็ม คือ $r = 3$ ตัวอย่างฟังก์ชัน staticInsertQ เพื่อใช้ในการนำข้อมูลเข้าไปเก็บในคิว โดยกำหนดให้ f แทนตัวชี้ต้นคิว, r แทนตัวชี้ท้ายคิว, q แทนขนาดของคิว และ item แทนข้อมูลที่จะนำไปเก็บในคิว

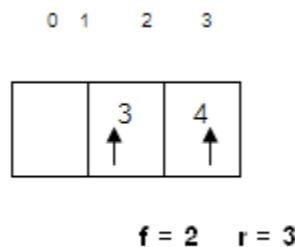
2) การนำสมาชิกออกจากคิว (deletion)

เป็นการนำข้อมูลที่เก็บอยู่ในคิวออกจากคิว โดยเมื่อทำการ deletion ข้อมูลนั้นออกจากคิวแล้ว จะต้องมีการจัดการให้ตัวชี้คิว f ซึ่งไปยังช่องหรือตำแหน่งต่อจากข้อมูลที่จะได้ทำการ deletion ไปแล้ว ส่วนตัวชี้คิว r ยังคงซึ่งไปยังช่องข้อมูลสุดท้ายเหมือนเดิม

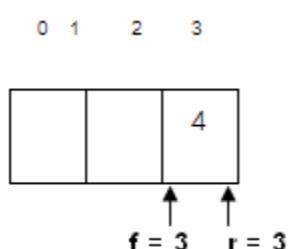
การ deletion จะทำการนำข้อมูลในส่วนของข้อมูลตัวแรกสุดที่เข้าสู่คิวออกไปทำงานตามต้องการ แต่การ deletion ข้อมูลจะไม่สามารถ deletion ข้อมูลออกจากคิวที่ว่างเปล่า หรือไม่มีข้อมูลได้ ($f = 0$) ถ้าเกิดกรณีเช่นนี้จะเกิดข้อผิดพลาดที่เรียกว่า underflow ขึ้น จะนั่นก่อนที่จะทำการ deletion ควรที่จะต้องมีการตรวจสอบว่าคิวว่างหรือไม่ เพื่อไม่ให้เกิดข้อผิดพลาดนี้ขึ้น

จากตัวอย่างข้างบน เมื่อทำการ deletion

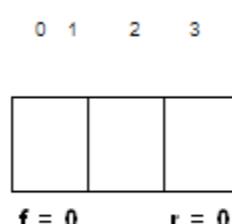
5. delete ()



6. delete ()



7. delete ()



หลังจากนี้จะทำ delete อีกไม่ได้ เนื่องจากคิวว่าง คือ $f = 0$ พิจารณาขั้นตอนทั้ง 7 ขั้นตอน สรุปได้ว่า

1. การเพิ่มข้อมูลเข้า ซึ่งต้องเข้าที่ท้ายคิว โดยการเพิ่มค่า r ที่เป็นตัวชี้สมาชิกตัวสุดท้ายของคิวอีก 1 เพื่อให้ชีenne ที่ถัดไป สำหรับเตรียมให้นำข้อมูลมาใช้

2. การนำข้อมูลออกจากคิว ทำที่ต้นคิว ซึ่ง f ชี้สมาชิกตัวแรกของคิวอยู่แล้ว ดังนั้น จึงนำข้อมูลที่ f ชี้อยู่ออกได้เลย หลังจากนั้นเปลี่ยนให้ f ไปชี้สมาชิกตัวถัดไป โดยการเพิ่มค่า f ขึ้นอีก 1

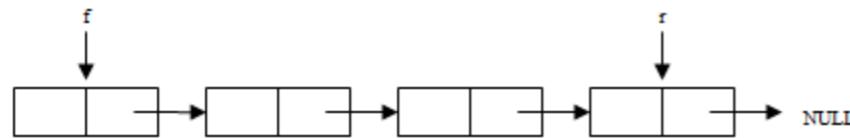
จากข้อสรุป 2 ข้อนี้ เป็นเพียงข้อสรุปที่เป็นกรณีทั่วๆ ไป แต่ยังมีกรณีพิเศษอีก 2 กรณี คือ

3. (ให้พิจารณาขั้นตอนที่ 1 และ 2) คือเมื่อเพิ่มข้อมูลเข้าคิวที่ว่างอยู่ต้องกำหนดค่า f ที่แต่เดิมชี้ที่ศูนย์ ให้ชี้ที่ 1 ด้วย เพื่อให้ f ชี้ที่สมาชิกแรกของคิว

4. (ให้พิจารณาขั้นตอนที่ 6 และ 7) คือเมื่อทำการนำข้อมูลออกจากคิวในขณะที่ในคิวนั้นมีสมาชิกเดียว หลังจากนำข้อมูลออกแล้วก็หมายความว่าคิวต้องว่าง ดังนั้นจึงต้องกำหนดค่าของ f ให้ไปชี้ที่ศูนย์ เช่นเดียวกันกับ r

2. การแทนที่คิวแบบไดนามิก

เช่นเดียวกันกับโครงสร้างสแตก การแทนที่คิวแบบไดนามิกจะใช้โครงสร้างแบบลิงค์ลิสต์เดียว โดยจะต้องสร้างลิงค์ลิสต์เดียวแบบมีตัวชี้ 2 ตัว กำกับอยู่ที่ต้นคิวและท้ายคิวดังภาพ



โดย f จะทำหน้าที่เป็นตัวชี้ที่ชี้ไปยังสมาชิกตัวแรกสุดในคิว และ r จะทำหน้าที่ชี้ไปยังสมาชิกตัวสุดท้ายในคิว และในกรณีที่มีการเพิ่มสมาชิกเข้าไปในคิว สมาชิกใหม่นี้จะถูกนำไปเสริมทางด้านหลัง (r ชี้อยู่) ส่วนกรณีที่มีการนำสมาชิกออกจากคิวจะนำสมาชิกออกจากทางด้านหน้าที่ตำแหน่ง f

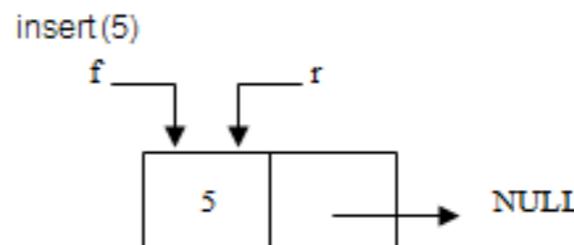
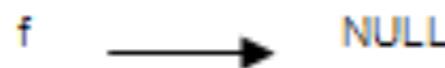
1) การนำสมาชิกใหม่เข้าไปในคิว (insert)

เช่นเดียวกับคิวแบบสแตติก การนำข้อมูลเข้าไปในคิวจะมี 2 กรณี คือ

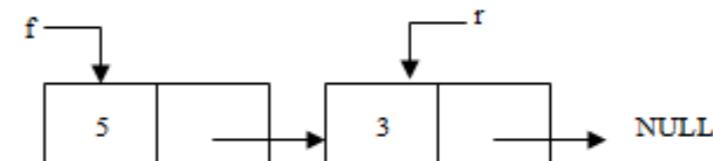
การนำข้อมูลเข้าไปในคิวว่าง โดยจะต้องดำเนินการให้พอยน์เตอร์ทั้ง 2 คือ ตัวชี้หัวคิว (f) และตัวชี้หางคิว (r) ชี้ไปยังตำแหน่งของข้อมูลแรก

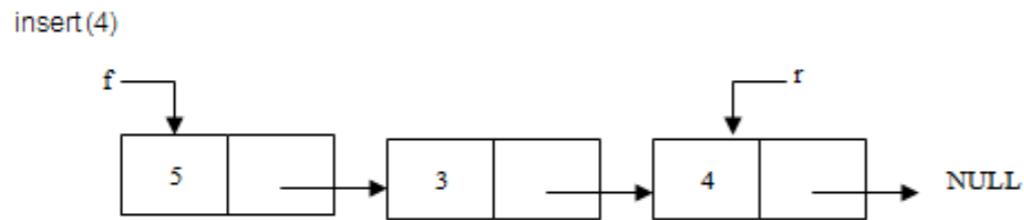
การนำข้อมูลเข้าไปในคิวต่อจากข้อมูลเดิม จะต้องจัดการให้ตัวชี้หางคิว ชี้ไปยังตำแหน่งของข้อมูลที่นำเข้าไป ส่วนตัวชี้หัวคิวยังคงชี้ไปยังตำแหน่งของข้อมูลที่นำเข้าไปเป็นข้อมูลแรก

ดังตัวอย่าง ในกรณีที่คิวว่าง ตัวชี้ f และ r จะมีค่าเป็น NULL



insert(3)





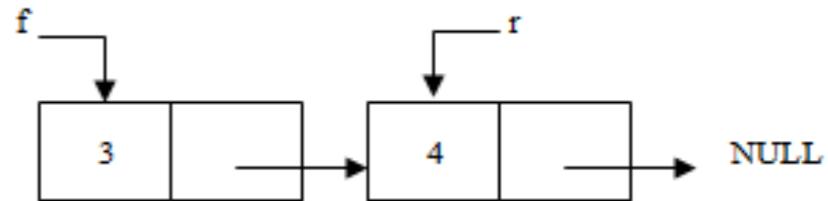
2) การนำข้อมูลออกจากคิว (deletion)

เป็นการนำข้อมูลที่เก็บอยู่ในคิวออกจากคิว โดยเมื่อทำการ deletion ข้อมูลนั้นออกจากคิวแล้ว จะต้องมีการจัดการให้ตัวชี้คิว f ซึ่งเป็นตัวแทนของโหนดที่อยู่ต่อจากโหนดที่จะทำการ deletion ไปแล้ว ส่วนตัวชี้คิว r ยังคงซึ่งเป็นตัวแทนของโหนดสุดท้ายเหมือนเดิม และจะต้องส่งโหนดที่ delete แล้วกลับคืนไป storage pool

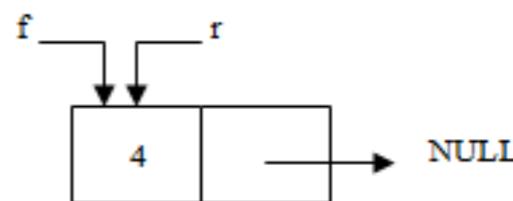
การ deletion จะทำการนำข้อมูลในส่วนของข้อมูลตัวแรกสุดที่เข้าสู่คิวออกไปทำงานตามต้องการ แต่การ deletion ข้อมูลจะไม่สามารถ deletion ข้อมูลออกจากคิวที่ว่างเปล่า หรือไม่มีข้อมูลได้ ($f = 0$) ถ้าเกิดกรณีเช่นนี้จะเกิดข้อผิดพลาดที่เรียกว่า underflow ขึ้น จะนั่นก่อนที่จะทำการ deletion ควรที่จะต้องมีการตรวจสอบว่าคิวว่างหรือไม่ เพื่อไม่ให้เกิดข้อผิดพลาดนี้ขึ้น

จากตัวอย่างข้างบน เมื่อทำการ deletion

5. delete ()



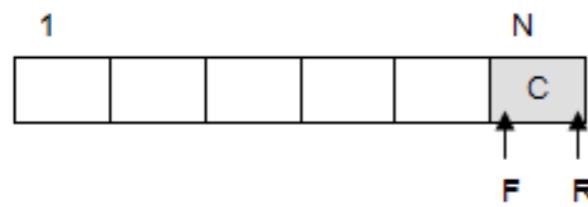
6. delete ()



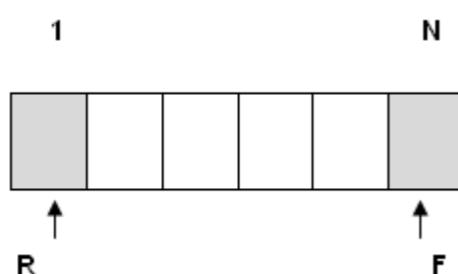
7. delete ()

$f \longrightarrow \text{NULL}$
 $r \longrightarrow \text{NULL}$

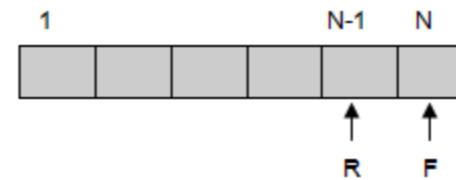
คิววงกลม (Circular Queue)



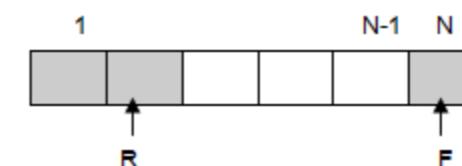
จากรูปข้างต้น ในคิวมีเพียงสมาชิกเดียว แต่ R ชี้เนื้อที่สุดท้ายของคิว ในลักษณะนี้ถ้าจะทำการเพิ่มข้อมูลเข้าในคิว อีกย่อไม่ได้ เนื่องจาก ถ้า $Q = R$ แล้วแสดงว่าคิวเต็ม แม้ว่ายังมีเนื้อที่เหลือด้านหน้าก็ตาม ทำให้การใช้เนื้อที่ไม่เต็ม ที่ทางแก้ไขก็คือยอมให้เนื้อที่ส่วนหน้าถูกใช้บรรจุข้อมูลได้ โดยให้ส่วนท้ายของคิวที่ตำแหน่ง $Q(N)$ ไปต่อกับส่วนหน้าที่ตำแหน่ง $Q(1)$ โครงสร้างคิวแบบนี้เรียกว่า เชอร์คูล่าคิว (Circular Queue) ในคิวแบบธรรมดาก็เมื่อ $R = N$ จะหมายความว่าคิวนั้นเต็ม แต่คิวแบบ Circular เมื่อ $R = N$ เราจะปรับให้ $R = 1$



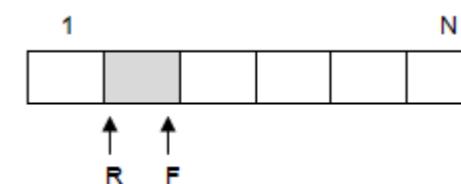
เชอร์คูล่าคิวนี้สามารถเพิ่มข้อมูลเข้าได้เรื่อย ๆ จนเป็นรูปข้างล่าง จึงจะไม่สามารถเพิ่มได้อีกซึ่งหมายถึงคิวเต็ม



ในการนิ่มนำข้อมูลออกจากเชอร์คูล่าคิว จุดที่ต้องพิจารณาคือขณะที่ F ชี้ที่ N (เนื้อที่สุดท้ายของคิว) โดยที่คิวมีสมาชิกมากกว่าหนึ่ง หรือ F กับ R ไม่ได้ชี้ตำแหน่งเดียวกัน



ในการนิ่มนำข้อมูลที่ F ออกแล้วต้องเซตค่า F ให้ชี้ที่สมาชิกตัวแรกของคิว คือ F ชี้ที่ตำแหน่ง 1 และถ้าก่อนนำข้อมูลออกโดยที่ในคิวมีสมาชิกเพียงสมาชิกเดียว ดังรูป หลังจากนำข้อมูลออกไปแล้ว ให้เซตค่า F และ R ให้เป็นคุณย์เพื่อแสดงว่าคิวว่าง



Section 2

การประยุกต์ใช้คิวจำลองแบบ

การจำลองแบบ (Simulation) หมายถึง การใช้ระบบหนึ่งเพื่อเลียนแบบพฤติกรรมของอีกรอบหนึ่ง ใช้งานเมื่อการทดลองด้วยระบบจริงๆ มีค่าใช้จ่ายสูง หรือเสี่ยงต่ออันตราย การจำลองแบบของคอมพิวเตอร์ จะใช้ขั้นตอนการทำงานของโปรแกรม เพื่อการเลียนแบบพฤติกรรมของระบบที่เราต้องการศึกษา

การจำลองแบบของระบบแบ่งกันใช้เวลา

ระบบคอมพิวเตอร์ที่มีการทำงานแบบแบ่งกันใช้เวลา



เป็นระบบที่มีผู้ใช้เครื่องคอมพิวเตอร์ร่วมกันในเวลาเดียวกัน โดยระบบมีหน่วยผลกระทบ (ซีพีयู) และหน่วยความจำหลัก

เพียงอย่างละ 1 เท่านั้น ผู้ใช้หลาย ๆ คนนี้ จะต้องมีการใช้หน่วยความจำหลักและหน่วยประมวลผลกลางร่วมกัน ซึ่งอนุญาติให้ผู้ใช้แต่ละคนประมวลผลโปรแกรม (ใช้ทรัพยากรของระบบ) ในเวลาหนึ่งๆ และก็จะให้ผู้ใช้คนต่อไปใช้จนกว่าจะวนกลับมา y ผู้ใช้คนแรกอีก วิธีการประมวลผลร่วมกันระหว่างผู้ใช้หลาย ๆ คน เราเรียกว่า ระบบแบ่งกันใช้เวลา (Time sharing) ซึ่งลักษณะการใช้ซีพีyu จะเป็นไปตามลำดับคือ “มาก่อนได้ก่อน” (first – come – first – serve) และมีลำดับการทำงานดังนี้

เมื่อโปรแกรมขอใช้เวลาซีพีyu โปรแกรมนั้นจะถูกนำ进来ต่อท้ายคิวประมวลผล

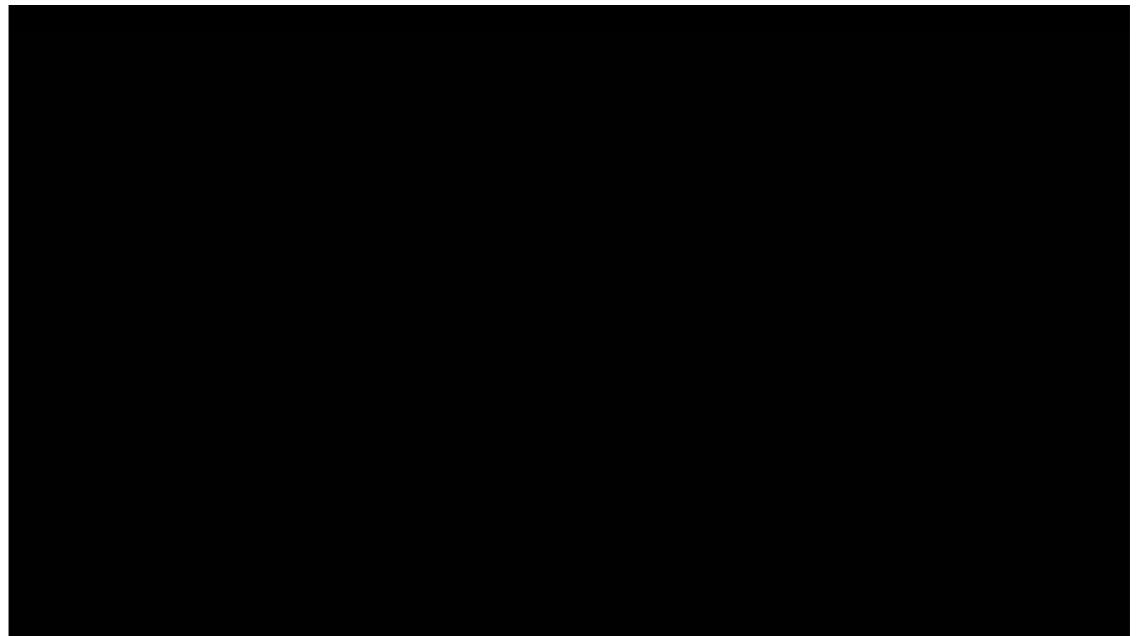
โปรแกรมที่อยู่ต้นคิวจะถูกส่งไปทำงาน และยังคงอยู่ที่ต้นคิวจนกว่าจะใช้ซีพีyu เสร็จ

เมื่อรันโปรแกรมทำงานเสร็จตามเวลาการขอใช้ซีพีyu ก็จะถูกนำออกจากคิว และจะไม่ถูกนำกลับมาอีก จนกว่าจะมีการขอใช้ซีพีyu ครั้งใหม่ (จึงจะกลับไปที่ข้อ 1. อีก)

การจำลองแบบสนามบิน

การเขียนโปรแกรมเพื่อจำลองแบบของสนามบิน จะใช้โครงสร้างข้อมูลแบบคิว แทนคิวของเครื่องบินที่รอขึ้นหรือรอลง แต่ตอนข้างเป็นโปรแกรมที่ซับซ้อน ดังสภาพความเป็นจริงที่ว่า สนามบินมีขนาดเล็กแต่มีเครื่องบินขึ้นลงจำนวนมาก มีทางวิ่ง (runway) เพียงทางเดียว ดังนั้น ณ เวลาใด ๆ เครื่องบิน จะต้องขึ้นหรือลงอย่าง โดยย่างหนึ่งเท่านั้น และเพียงเครื่องเดียวด้วย ในเวลาที่เครื่องบินซึ่งพร้อมจะขึ้นหรือลงมาถึงสนามบิน สนามบินนั้นอาจจะว่างหรือมีเครื่องบินอื่นกำลังขึ้นหรือลงอยู่ก็ได้ และอาจจะมีเครื่องบินหลายลำที่รอขึ้นและรอลง จึงมีคิว 2 คิวเกิดขึ้น คือ คิวขึ้น (takeoff) และคิวลง (landing) ในการรอนั่นบนพื้นจะดีกว่าบนอากาศ ดังนั้นจึงให้เครื่องบินขึ้นได้ก็ต่อเมื่อไม่มีเครื่องบินลง หลักจากได้รับสัญญาณร้องขอจากเครื่องบินลำใหม่เพื่อจะลงหรือขึ้น โปรแกรมการจำลองแบบจะให้บริการเครื่องบินที่อยู่ในตำแหน่งหัวคิวของคิวลงก่อน และถ้าคิวลงว่าง จึงอนุญาตให้เครื่องบินในคิวขึ้นขึ้นได้ โปรแกรมจำลองแบบนี้สามารถจะทำงานได้ตลอดเวลา

Movie 6.1 แสดงการจำลองสนามบิน

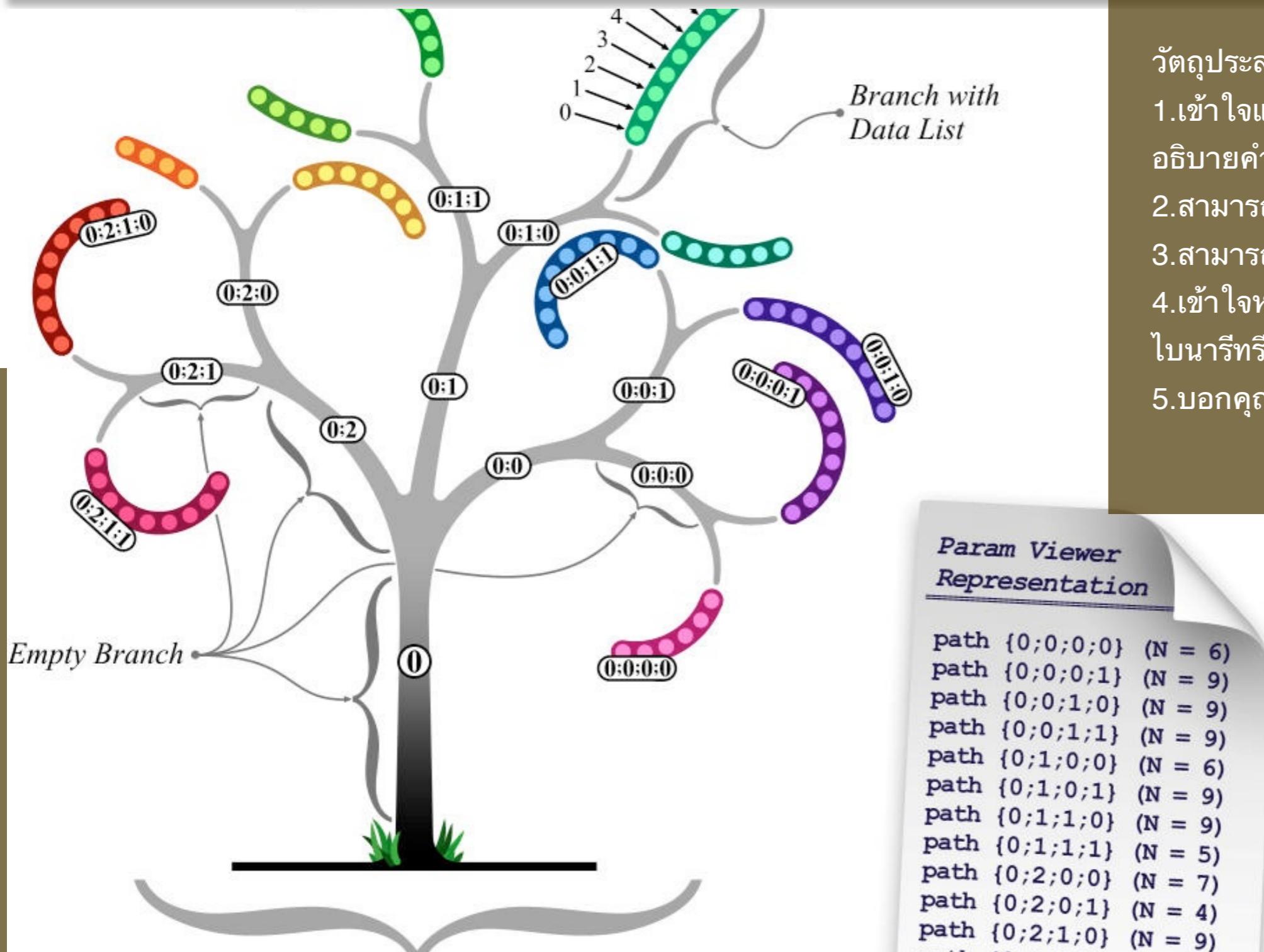


Section 3

แบบฝึกหัด

1. สมมติโครงสร้างคิวมีขนาด 5 และมีการนำข้อมูลมาดำเนินการกับโครงสร้างข้อมูล ดังนี้
 - 1.1 Qinsert (A)
 - 1.2 Qinsert (B)
 - 1.3 Qinsert (C)
 - 1.4 Qdelete (Item)
 - 1.5 Qdelete (Item)
 - 1.6 Qinsert (D)
 - 1.7 Qinsert (E)
 - 1.8 Qinsert (F)
 - 1.9 Qinsert (G)
 - 1.10 Qdelete (Item)
 - 1.11 Qinsert (H)
 - 1.12 Qinsert (I)
2. จงวาดรูปจากข้อมูลในข้อแรก โดยใช้คิวแบบวงกลม
3. จงยกตัวอย่าง การใช้คิวในชีวิตประจำวัน หรือในระบบคอมพิวเตอร์ มา 5 ตัวอย่าง
4. คิวเป็นโครงสร้างแบบใด และมีความแตกต่างจากสแตกอย่างไรจオธิบาย

Tree Structure



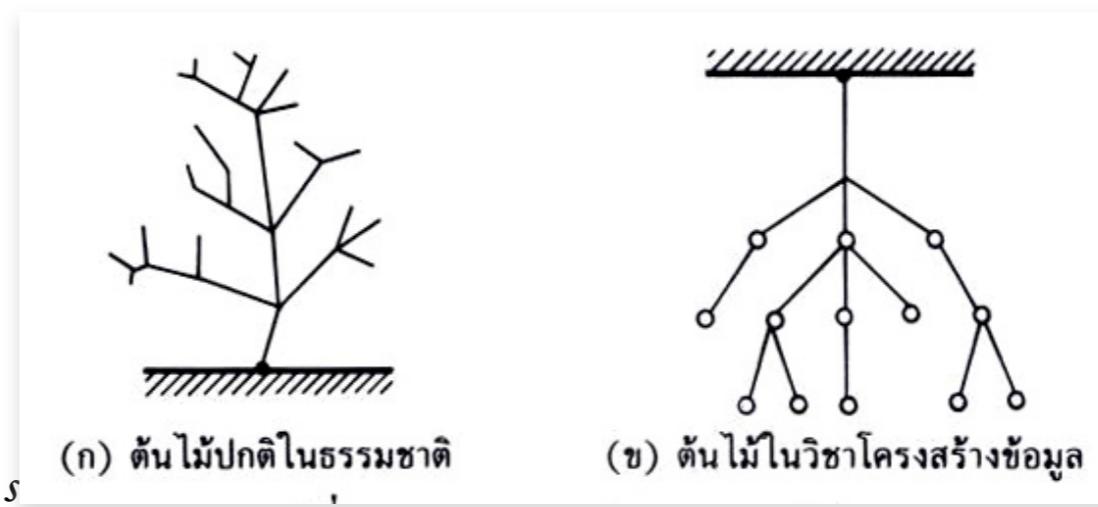
วัตถุประสงค์

- เข้าใจแนวคิดและหลักการของทรี รวมถึง อธิบายคำศัพท์พื้นฐานทรีได้
- สามารถอภิคุณสมบติของไบนารีทรีได้
- สามารถท่องเข้าไปในไบนารีทรีได้
- เข้าใจหลักการแปลงเจเนอร์ลทรีมาเป็น ไบนารีทรีได้
- บอกคุณสมบติของไบนารีเสริชทรีได้

นิยามโครงสร้างข้อมูลทรี



ทรี(Tree)เป็นโครงสร้างที่มีความสัมพันธ์ในลักษณะลำดับชั้น สมาชิกแต่ละโหนดล้วนแต่มีความสัมพันธ์กัน ในลักษณะเหมือนครอบครัวเดียวกัน โดยมีโหนดพ่อซึ่งอยู่ระดับที่หนึ่อกว่ามีเส้นเชื่อม โยงจากโหนดพ่อไปยังโหนดที่อยู่ระดับต่ำกว่าที่เรียกว่า โหนดลูก



Section 1

องค์ประกอบของทรี

โหนด(Node) ที่ใช้สำหรับบรรจุข้อมูล โดยจะมีกิ่งซึ่งเป็นเส้นที่โยงโหนดเข้าด้วยกันที่

เรียกบранช์(Branch) จำนวนของบранช์ที่สัมพันธ์กับโหนดเรียกว่า

ดีกรี(Degree) และถ้าหากทรีนั้นเป็นทรีที่ไม่ใช่ทรีว่าง โหนดแรกจะเรียกว่า

ราก(Root) โดยโหนดทุกๆ โหนดยกเว้นราก โหนดจะมีเพียง 1 Predecessor ในขณะที่ Successor อาจเป็น 0 หรือ 1 หรือมากกว่า 1 ก็ได้ สำหรับ

Leaf ก็คือโหนดใบที่ไม่มีบранช์เชื่อม โยงไปยังโหนดตัวถัดไปหรือโหนดที่ไม่มีตัวตามหลังหรือ Successor นั้นเอง ในขณะที่โหนด

พ่อ(Parent) จะมีโหนดตามหลังหรือมีโหนด

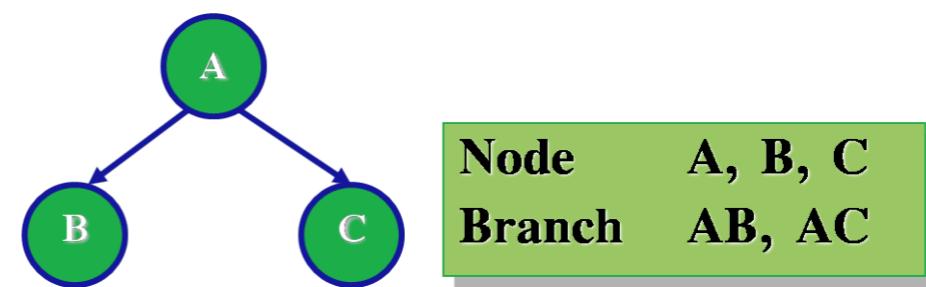
ลูก(Child) ต่อท้าย โหนดลูกตั้งแต่สอง โหนดหรือมากกว่า ที่มาจากการพ่อเดียวกันจะเรียกว่า โหนด

พี่น้อง(Siblings) โหนดต่างๆภายในทรีจะอยู่ในระดับที่ต่างกัน โดยเริ่มต้นจากราก โหนดซึ่งถือเป็นระดับแรกสุด(Level 0) ส่วนลูกๆของราก โหนดก็คือระดับที่ 1 (Level 1) และลูกๆของโหนดในระดับที่ 1 ก็จะอยู่ในระดับที่ 2 (Level 2) ซึ่งจะเพิ่มระดับไปเรื่อยๆ เมื่อลูกหลานเพิ่มขึ้น

คัพท์ต่าง ๆ เกี่ยวกับ Tree

1. Node ใช้เก็บข้อมูล

2. Branch ใช้เชื่อม Node เข้าด้วยกัน



3. Degree หมายถึง จำนวน Branch ที่สัมพันธ์กับ Node แบ่งเป็น

3.1 Indegree หมายถึง Branch เข้าหา Node

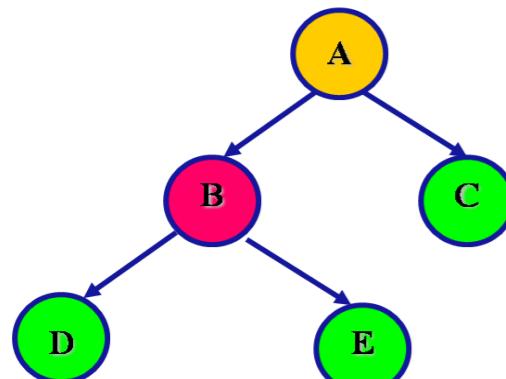
3.2 Outdegree หมายถึง Branch ที่ออกจาก Node

Node A มี Degree เท่ากับ 2
Indegree = 0
Outdegree = 2

4. Root หมายถึง Node แรกของ Tree
5. Leaf หรือ External node หมายถึง Node ที่มี Out degree เท่ากับ 0

Leaf B, C

6. Internal node หมายถึง Node ที่ไม่ใช่ Root และ Leaf



7. Parent หมายถึง Node ที่มี Outdegree

Parent A, B

8. Child หมายถึง Node ที่มี Indegree

Child B, C, D, E

9. Sibling หมายถึง Node ที่มี Parent เดียวกัน

Sibling {B, C}, {D, E}

10. Ancestor หมายถึง ทุก Node ในเส้นทางจาก Root ไปยัง Node ที่ต้องการ

Ancestor ของ E A, B

11. Descendent หมายถึง ทุก Node ในเส้นทาง จาก Node ที่กำหนดไปจนถึง Leaf

Descendent ของ A

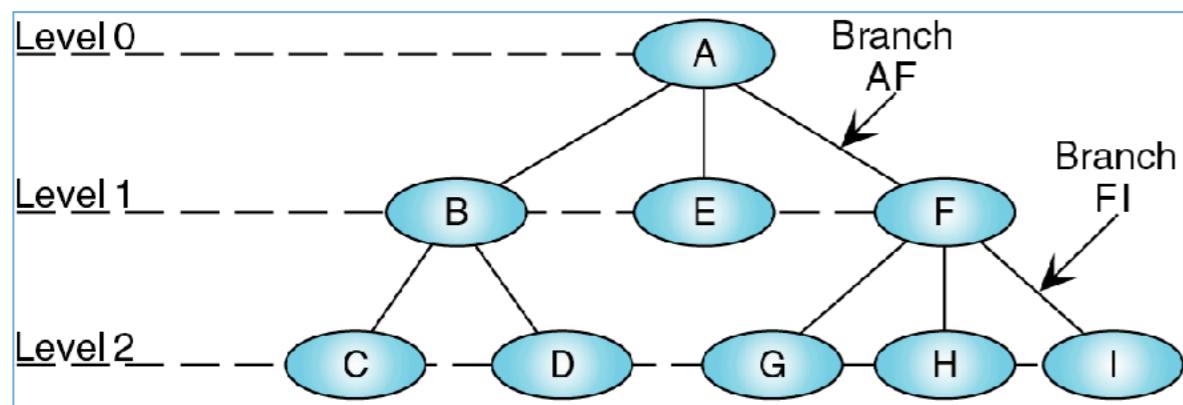
B, C, D, E

Descendent ของ B

D, E

12. Level หมายถึง ระยะทางจาก Root

13. Height ของ Tree หมายถึง Level สูงสุด ของ Leaf บวกด้วย 1

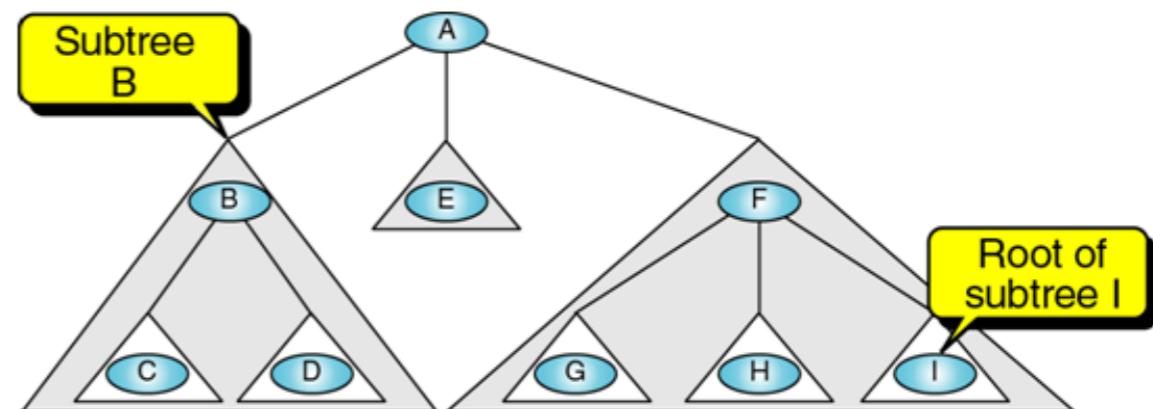


14. Subtree หมายถึง โครงสร้างที่เชื่อมต่อกันภายใต้ Root โดย

Node แรกของ Subtree จะเป็น Root ของ Subtree นั้น
และใช้เป็นชื่อเรียก Subtree

Subtree สามารถแบ่งย่อยเป็น Subtree ได้อีกจนกว่าจะ
Empty

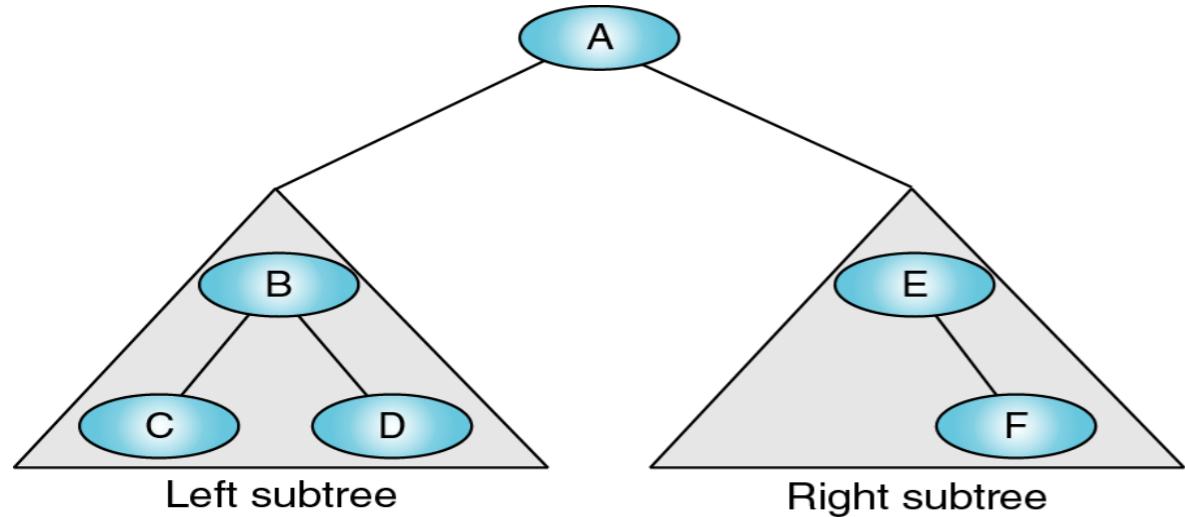
Subtree หมายถึง โครงสร้างที่เชื่อมต่อกันภายใต้ Root โดย Node แรกของ Subtree จะเป็น Root ของ Subtree นั้น และใช้เป็นชื่อเรียก Subtree Subtree สามารถแบ่งย่อยเป็น Subtree ได้อีกจนกว่าจะ Empty



Section 2

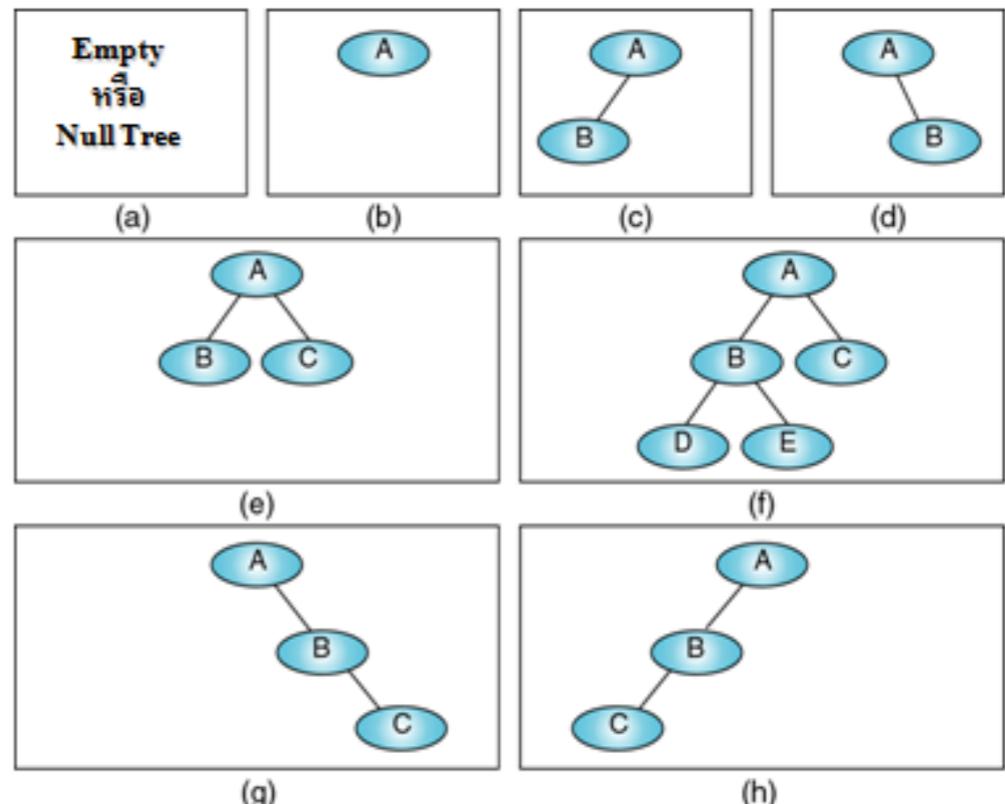
ไบนารีทรี (Binary Tree)

หมายถึง ต้นไม้ที่แต่ละ Node มี Subtree ≤ 2 หรือ
Out degree ≤ 2



ภาพแสดงโครงสร้างของไบนารีทรี

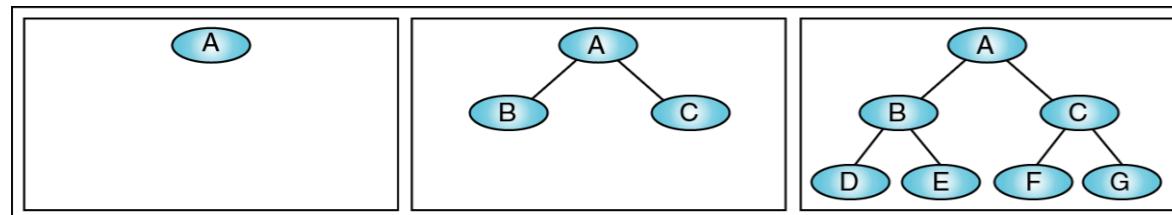
ตัวอย่าง Binary Tree



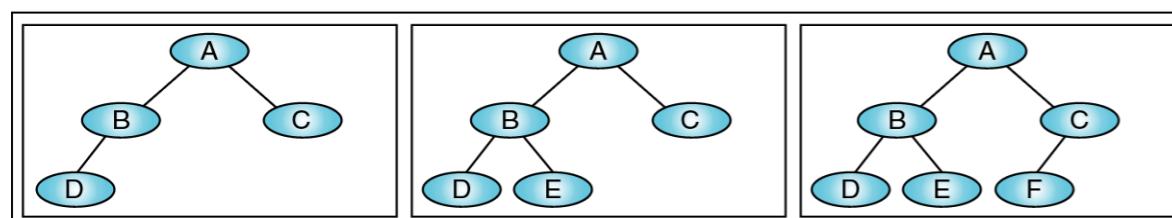
ในอาร์ทีแบบสมบูรณ์และเกือบสมบูรณ์

Complete Binary Tree เป็นทรีที่มีชั้นทรีด้านซ้ายและชั้นทรีด้านขวาเท่ากับ

Nearly Complete Binary Tree เป็นทรีที่มีโหนดเต็มทุกโหนด ยกเว้นใน Level สุดท้ายที่มีโหนดเฉพาะทางด้านซ้าย



(a) Complete trees (at levels 0, 1, and 2)

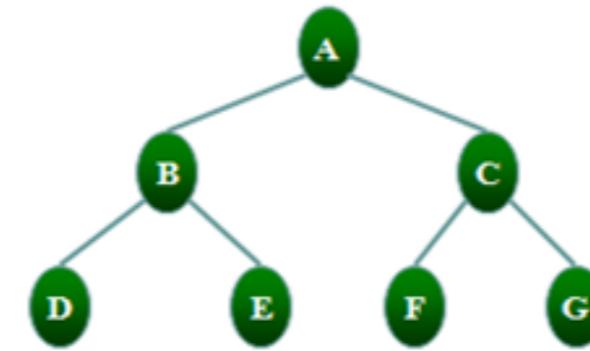


(b) Nearly complete trees (at level 2)

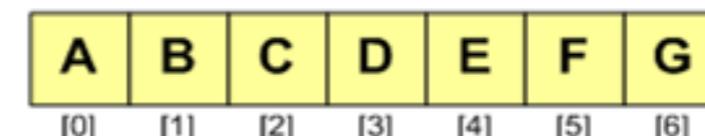
การแทนใบาร์ทีในหน่วยความจำ (Binary Tree Representations)

1. การแทนใบาร์ทีด้วยอาร์เรย์

รูปแบบของใบาร์ทีแบบสมบูรณ์

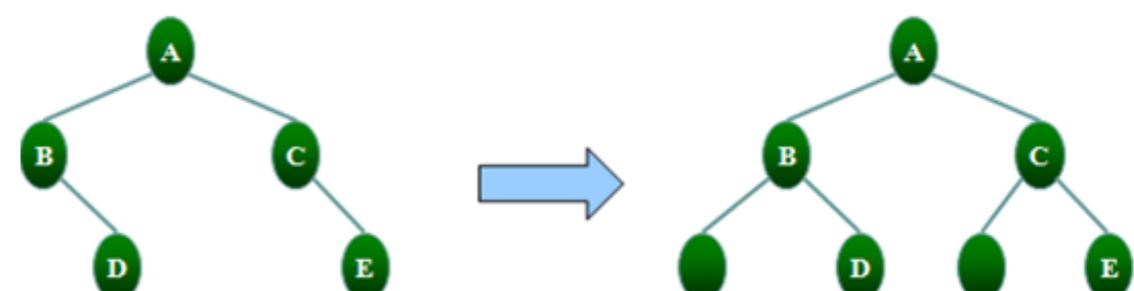


ในอาร์ทีแบบสมบูรณ์

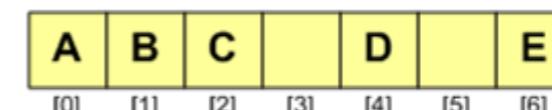


อาร์เรย์

รูปแบบของใบาร์ทีแบบไม่สมบูรณ์หรือเกือบสมบูรณ์

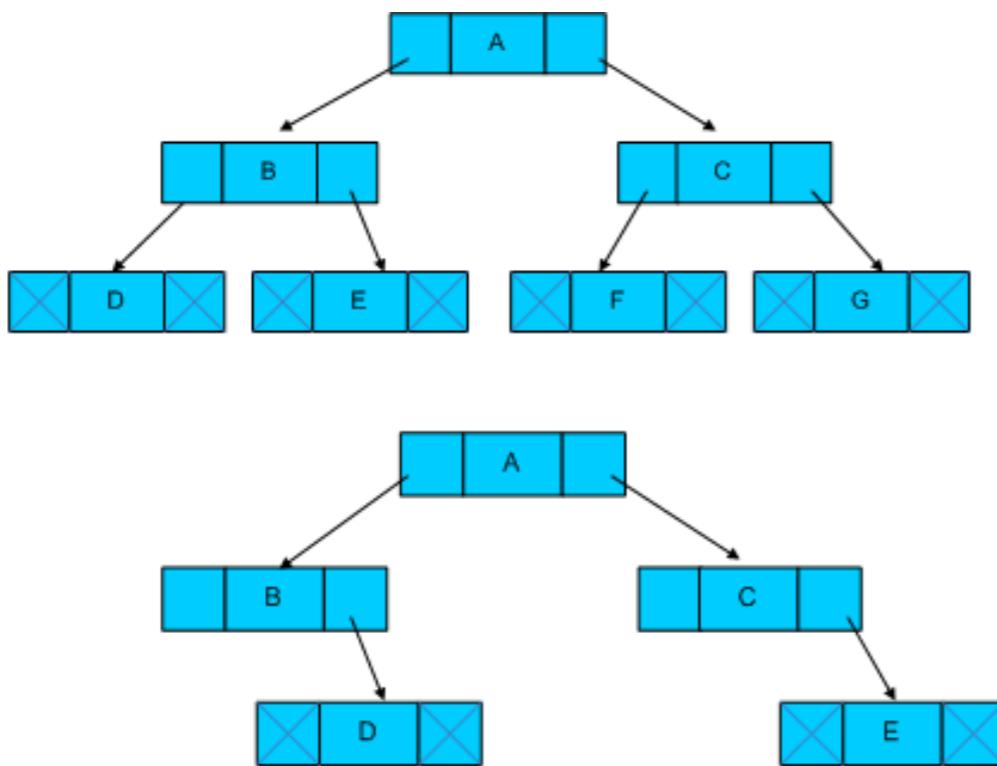


การแปลงใบาร์ทีแบบไม่สมบูรณ์ให้เป็นสมบูรณ์ให้ด้วยการเพิ่มโหนดว่างให้เต็ม



อาร์เรย์

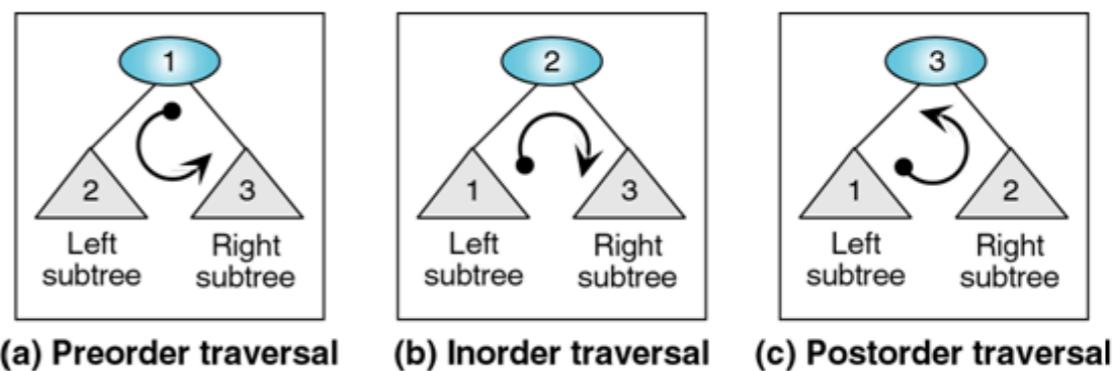
2. การแทนใบนาทรีด้วยลิงก์ลิสต์



ภาพแสดงการแทนใบนาทรีด้วยลิงก์ลิสต์

การท่องเข้าไปในใบนาทรี Binary Tree Traversal

1. Dept-first เป็นการท่องเข้าไปทรีด้วยการเดินทางผ่านจากราก โหนดลงไปยังโหนดลูก แบ่งเป็นทั้งหมด 6 แบบแต่มี 3 แบบที่นิยมใช้ได้แก่



NLR

LNR

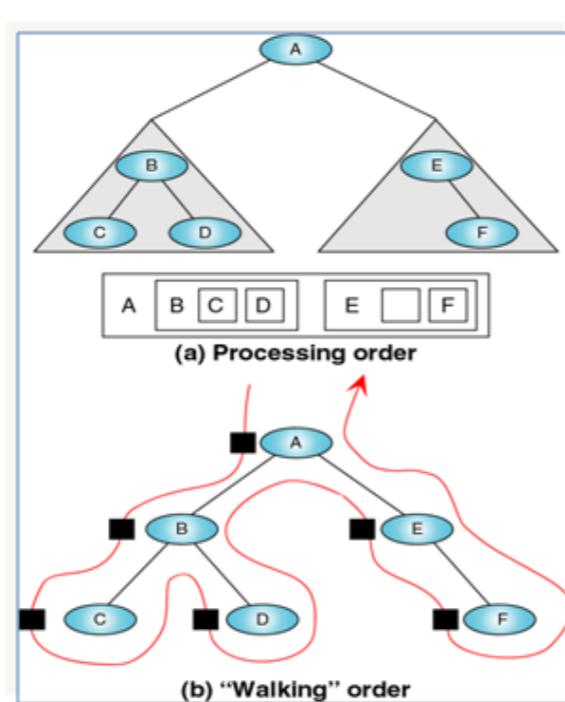
LRN

Preorder Traversal

N L R

```
algorithm preOrder (val root <node pointer>)
```

- 1 if (root is not null)
 - 1 process(root)
 - 2 preOrder(root->leftSubtree)
 - 3 preOrder(root->rightSubtree)
- 2 return

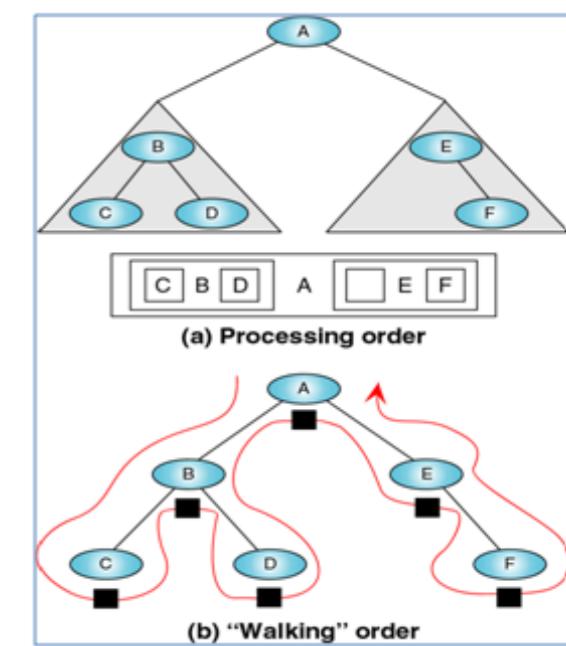


Inorder Traversal

L N R

```
algorithm inOrder (val root <node pointer>)
```

- 1 if (root is not null)
 - 1 inOrder(root->leftSubtree)
 - 2 process(root)
 - 3 inOrder(root->rightSubtree)
- 2 return

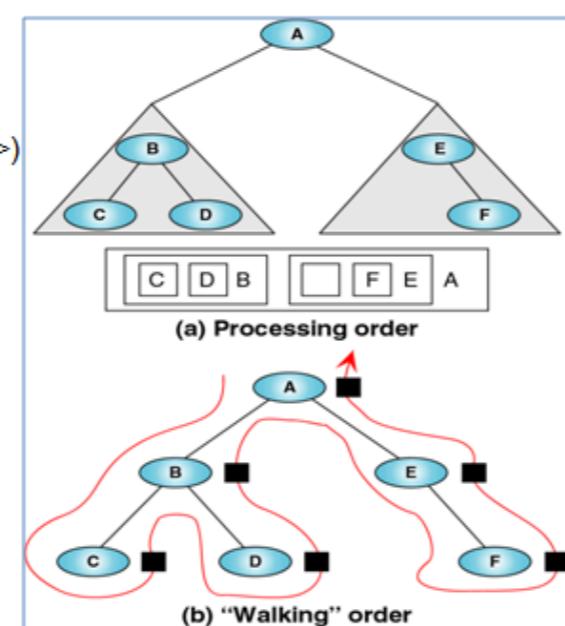


Postorder Traversal

L R N

```
algorithm postOrder (val root <node pointer>)
```

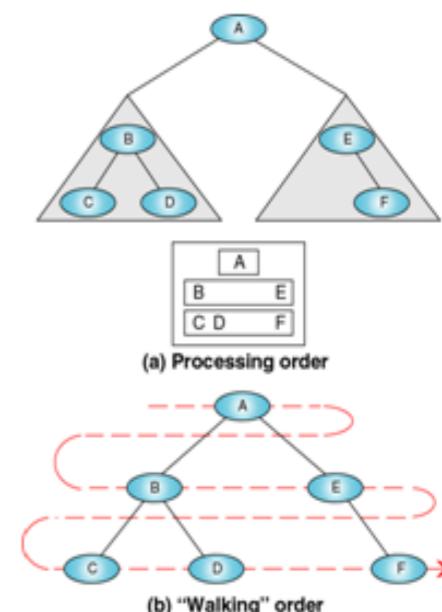
- 1 if (root is not null)
 - 1 postOrder(root->leftSubtree)
 - 2 postOrder(root->rightSubtree)
 - 3 process(root)
- 2 return



2. Breath-first ประมวลผลทีละ Level จากบนลงล่าง

```
algorithm breathFirst (val root <node pointer>)
```

- 1 p = root
- 2 while (p not null)
 - 1 process(p)
 - 2 if (p->left not null)
 - 1 enqueue(p->left)
 - 3 if (p->right not null)
 - 1 enqueue(p->right)
 - 4 if (not emptyQueue)
 - 1 dequeue(p)
 - else
 - 1 p = null
- 3 return



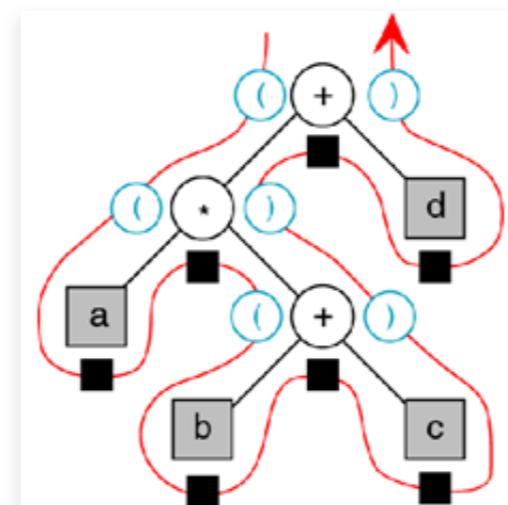
เอกสารเพรสชันทรี Binary Tree Application

คือ ไบนาリทรีซึ่งแทนนิพจน์ ซึ่งมีคุณสมบัติ ดังนี้

1. ตัวถูกดำเนินการหรือโอเปอเรตน์จะเก็บไว้ที่โหนดใบ (Leaf Node)
2. ตัวดำเนินการหรือโอเปอเรเตอร์จะเก็บไว้ที่รากโหนด หรือโหนดภายใน (Internal Nodes) ที่ไม่ใช่โหนดใบ
3. ชับทรี ในที่นี้ก็คือนิพจน์ย่อย โดยจะมีรากโหนดเป็นโอเปอเรเตอร์

การท่องใน Expression Tree

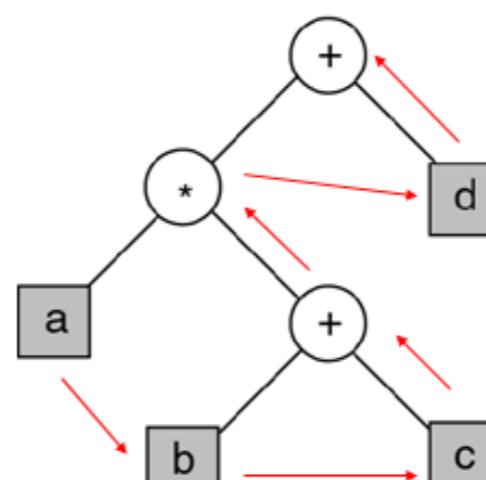
Inorder Traversal



$((a * (b + c)) + d)$

```
1 if (root is not null)
  1 if root->token is operand
    1 print(root->token)
  else
    1 print(open parenthesis)
    2 infix(root->leftSubtree)
    3 print(root->token)
    4 infix(root->rightSubtree)
    5 print(close parenthesis)
  2 return
```

Postorder Traversal

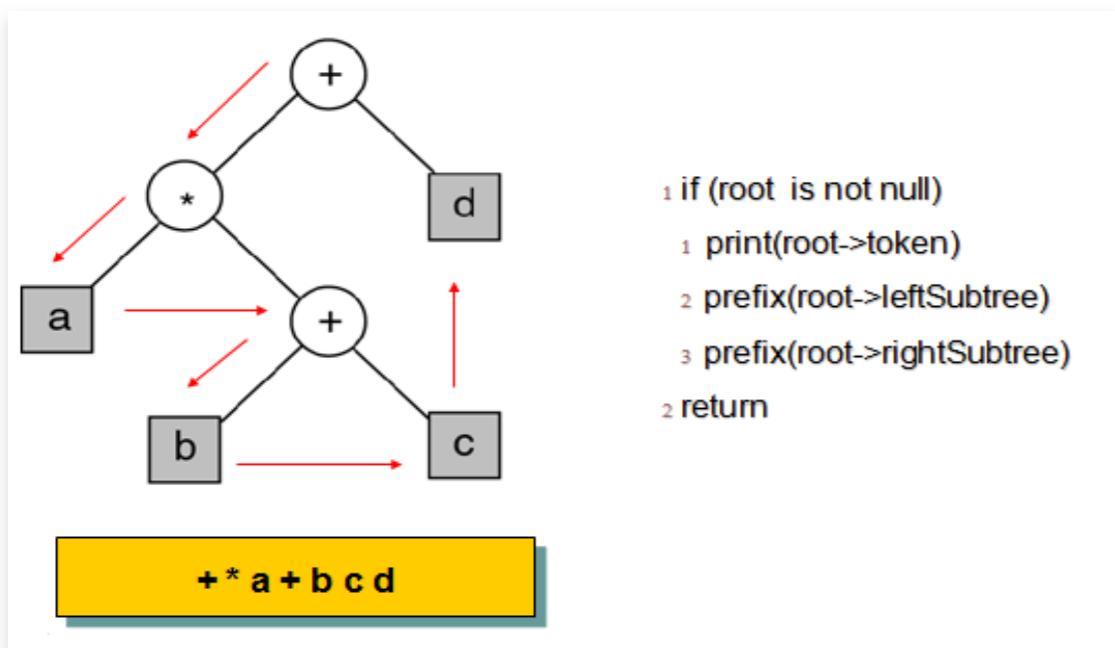


a b c + * d +

```
1 if (root is not null)
  1 postfix(root->leftSubtree)
  2 postfix(root->rightSubtree)
  3 print(root->token)
  2 return
```



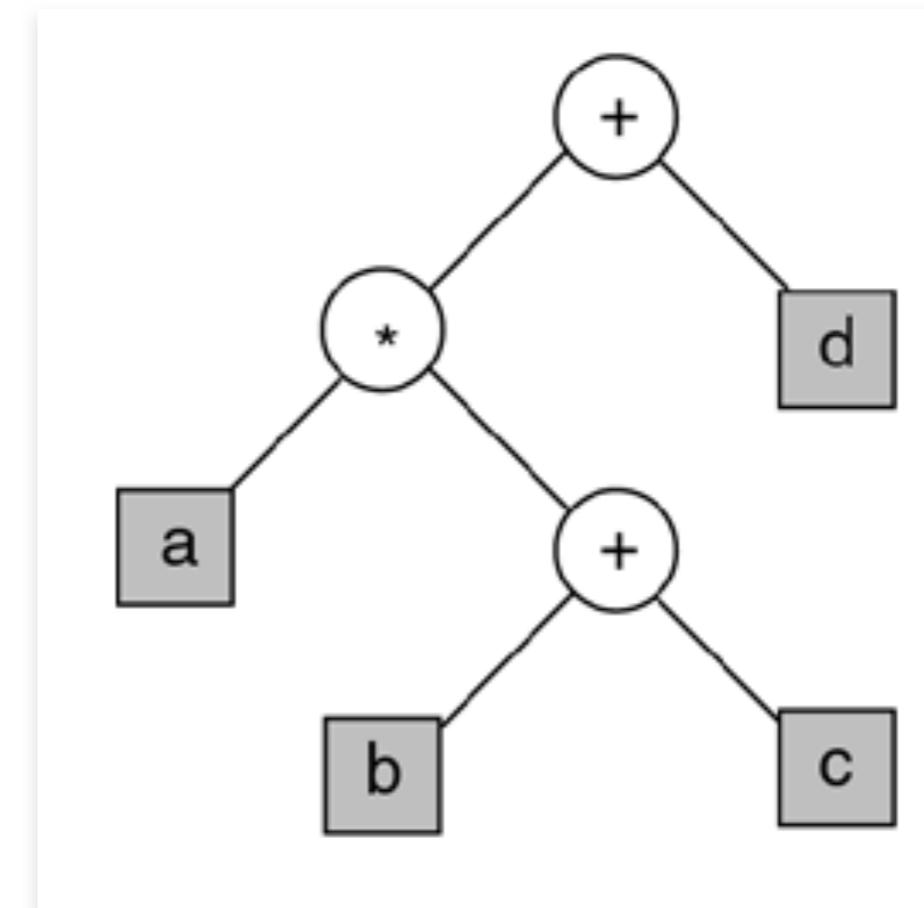
Preorder Traversal



การสร้าง Expression Tree จาก Postfix Expression

- ❖ พิจารณาทีละ Token จนหมด
- ❖ ถ้า Token เป็น Operand สร้าง Node แล้ว Push ลง Stack
- ❖ ถ้า Token เป็น Operator Pop ขึ้นมา 2 ตัวเชื่อมเป็น Tree โดยใช้ Operator แล้ว Push ลง Stack

ตัวอย่าง จากนิพจน์ Postfix **a b c + * d +** จะได้ Expression Tree ดังนี้



Section 3

เจเนอรัลทรี (General Tree)

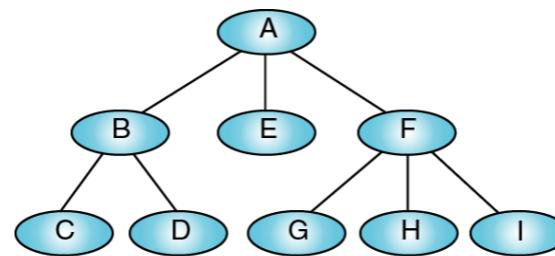
หมายถึง Tree ที่สามารถมี Out degree ได้ไม่จำกัดจำนวน

การแปลง General Tree เป็น Binary Tree

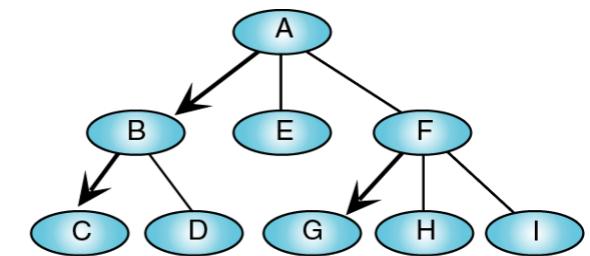
มี 3 ขั้นตอน ดังนี้

1. ระบุ Child ที่อยู่ทางซ้ายสุด
2. เชื่อม Sibling เข้าด้วยกัน
3. ลบ Branch ที่ไม่ต้องการ

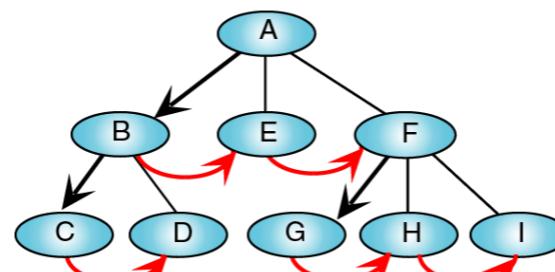
ตัวอย่าง การแปลง General Tree เป็น Binary Tree



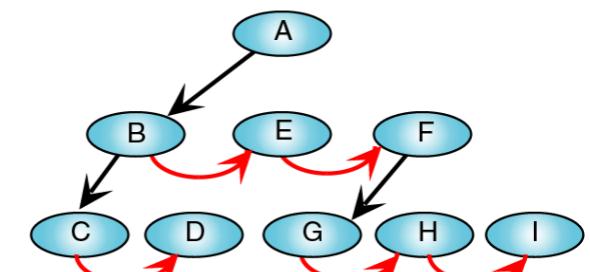
(a) The general tree



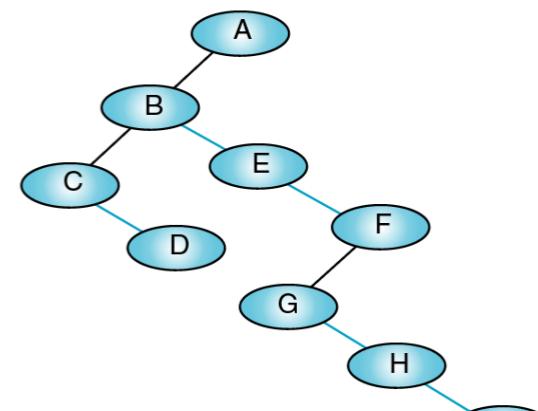
(b) Identify leftmost children



(c) Connect siblings



(d) Delete unneeded branches



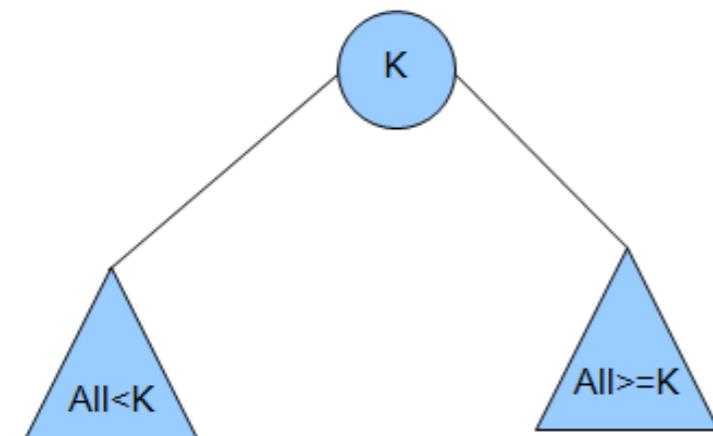
(e) The resulting binary tree

Section 4

ไบนารีเสิร์ชทรี Binary Search Tree : BST

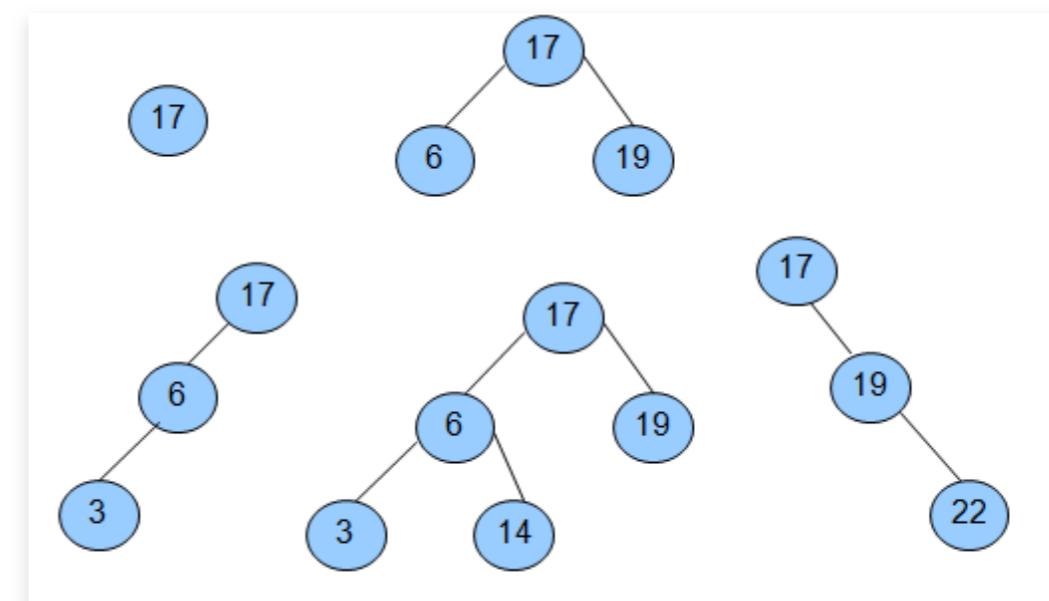
เป็นการนำไบนารีทรีมาประยุกต์ใช้งานเพื่อการค้นหาข้อมูล ไบนารีเสิร์ชทรีประกอบด้วยคุณสมบัติทุกๆ โนนดในชั้บที่ด้านซ้ายต้องมีค่าน้อยกว่าราก โนนดทุกๆ โนนด ในชั้บที่ด้านขวาต้องมีค่ามากกว่าหรือเท่ากับราก โนนดแต่ละชั้บทรีจะต้องเป็นไบนารีเสิร์ชทรี

ในไบนารีเสิร์ชทรี คีย์ที่บรรจุอยู่ในชั้บทรีด้านซ้ายจะมีค่าน้อยกว่าราก และคีย์ที่บรรจุอยู่ในชั้บทรีด้านขวาจะมีค่ามากกว่าหรือเท่ากับราก โนนด

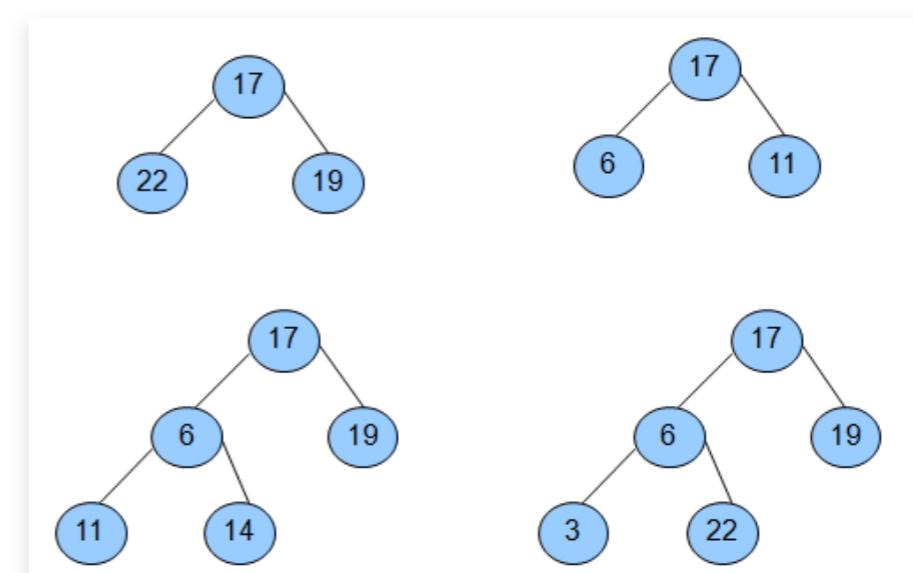


ภาพแสดงโครงสร้างไบนารีเสิร์ชทรี

ตัวอย่าง BST ที่ถูกต้อง

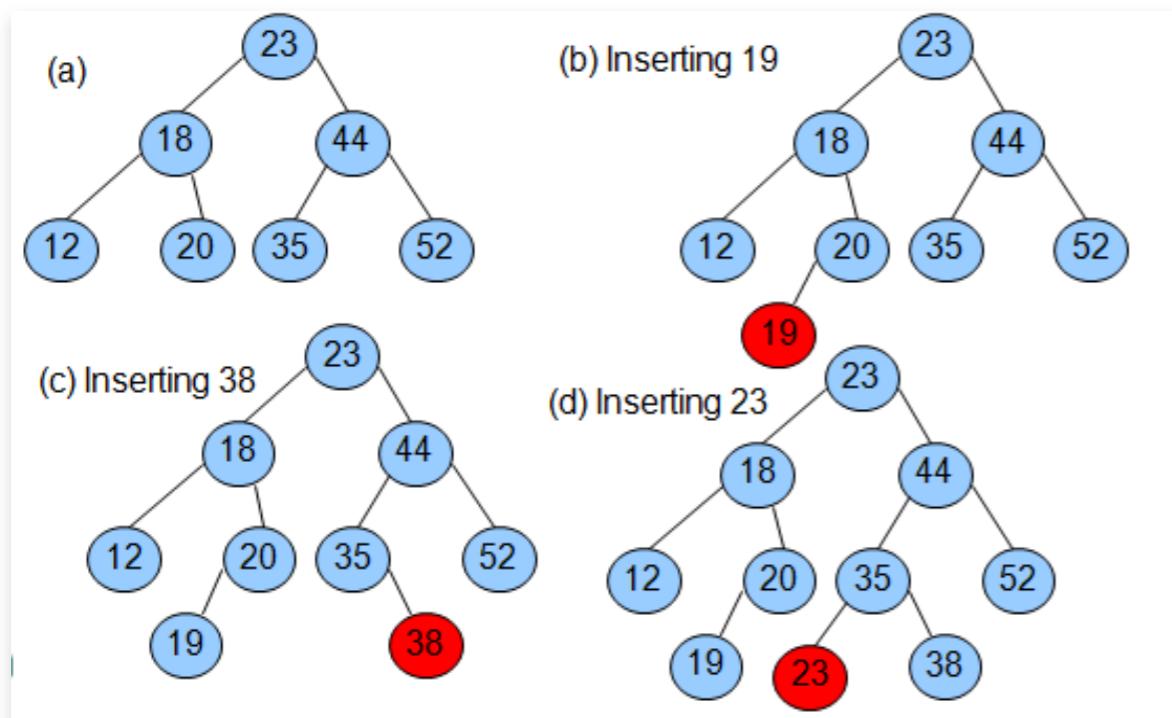


ตัวอย่าง BST ที่ผิด



การแทรกโหนดในไบนาเรียลซ์ทรี (Insertion)

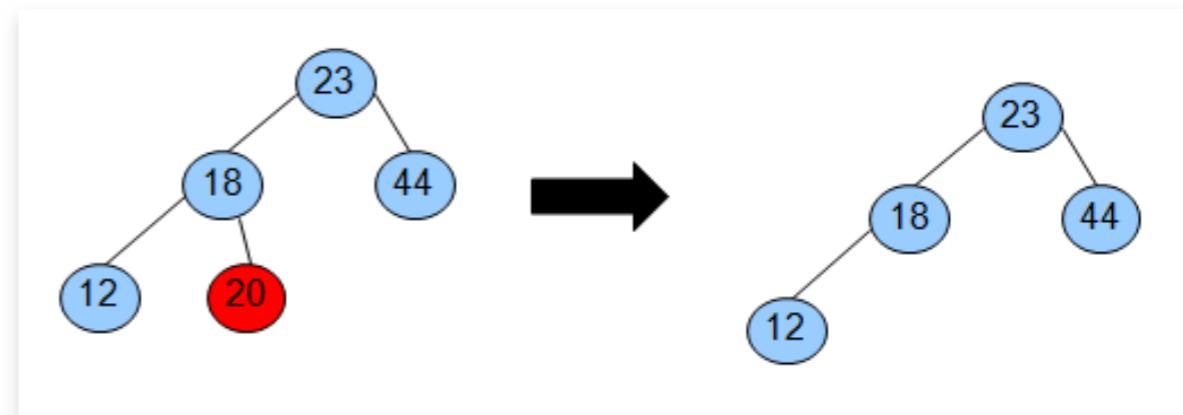
ให้ใช้หลักการคุณสมบัติของไบนาเรียลซ์ทรีในการแทรกโหนด



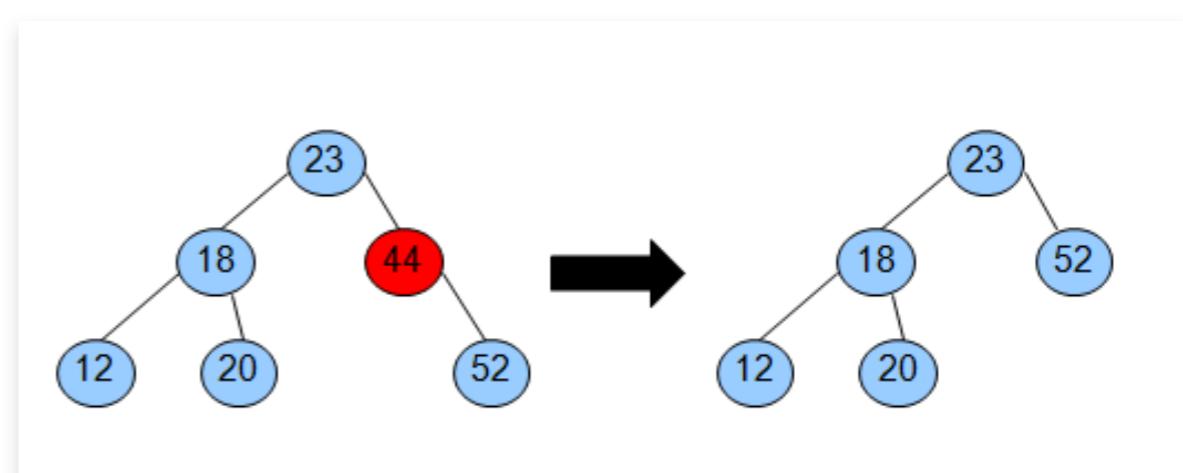
การลบโหนดในไบนาเรียลซ์ทรี (Deletion)

มี 4 ประการ

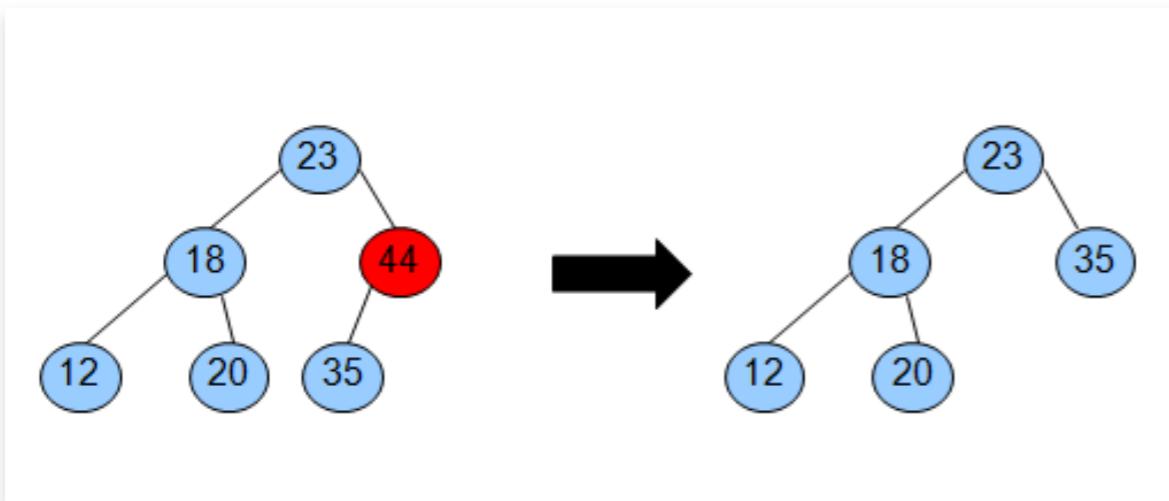
- ❖ กรณีโหนดที่ต้องการลบไม่มีลูก



- ❖ กรณีโหนดที่ต้องการลบมีเฉพาะชั้บทรีด้านขวา

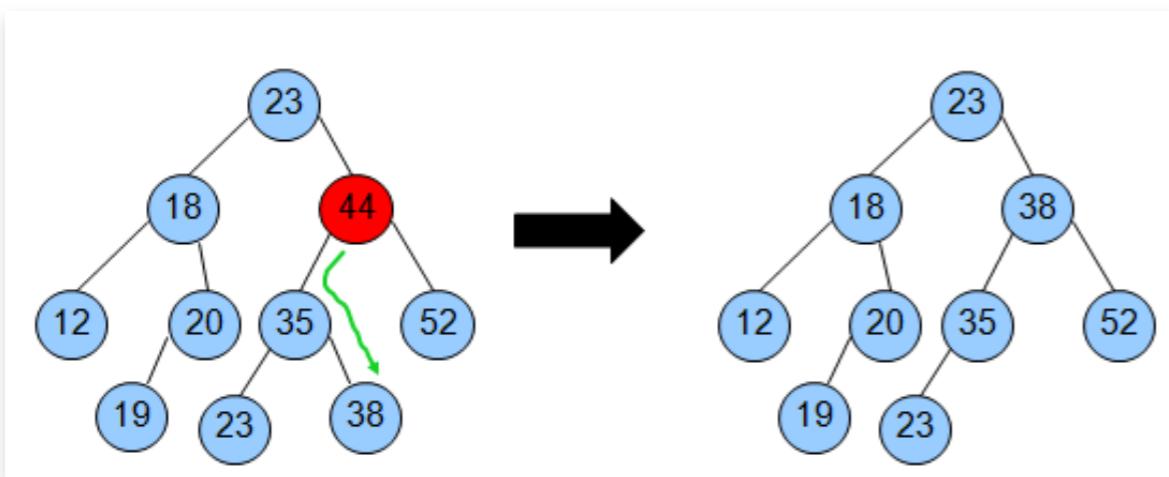


- ❖ กรณีโหนดที่ต้องการลบมีเฉพาะชั้บทรีด้านซ้าย

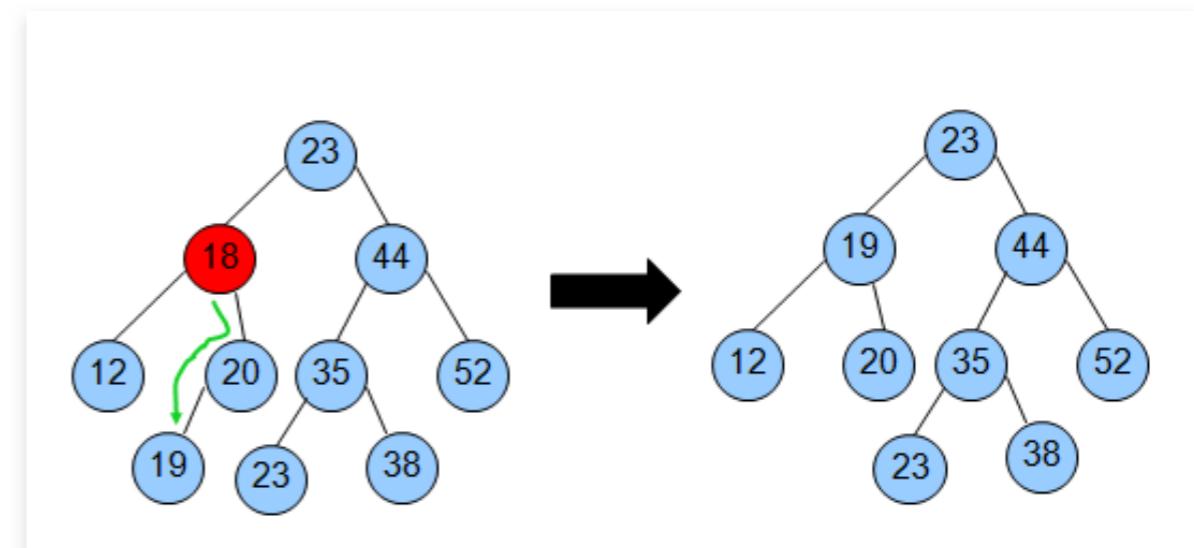


กรณีโหนดที่ต้องการลบมีสองชั้บที่ มีอยู่ 2 วิธี

- ❖ หาโหนดที่มีค่ามากที่สุดในชั้บที่ด้านซ้ายของโหนดที่ต้องการลบแล้วนำมาแทนที่



- ❖ หาโหนดที่มีค่าน้อยที่สุดในชั้บที่ด้านขวาของโหนดที่ต้องการลบแล้วนำมาแทนที่

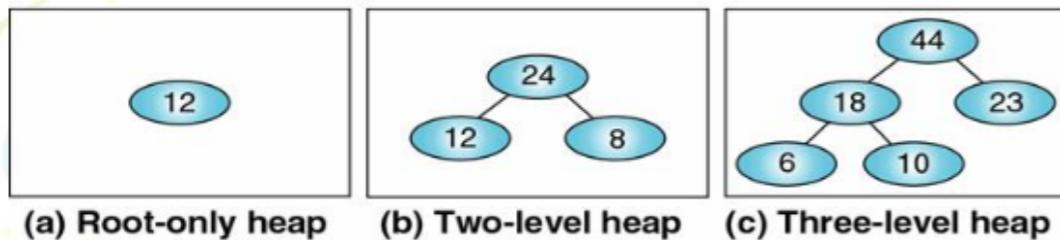


Section 5

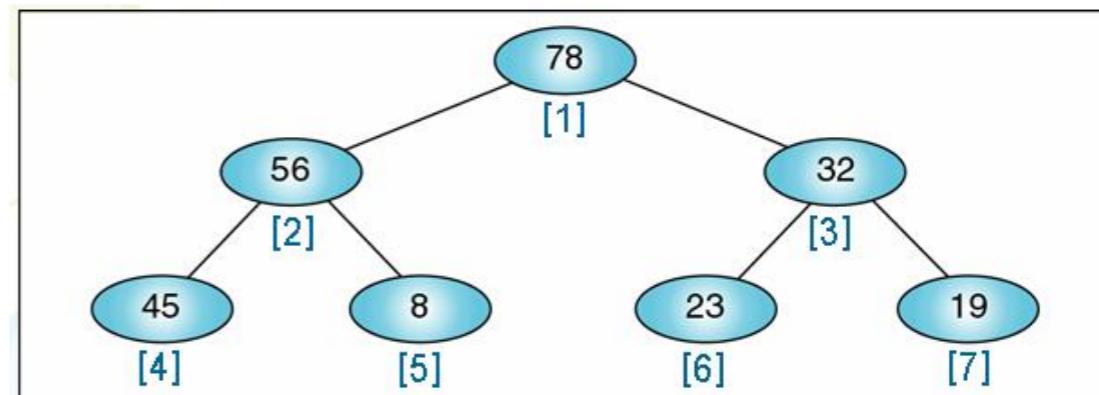
ฮีพทรี (Heap Tree)

โครงสร้างของทรีแบบฮีพนั้นก็คือ ในอารีทรีที่ชั้บทรีด้านซ้ายและด้านขวาจะมีค่าน้อยกว่าหรือเท่ากับ \leq โหนดพ่อ และรูดโหนดของฮีพทรีจะรับประกำบได้ว่ามีค่ามากที่สุด ในทรี

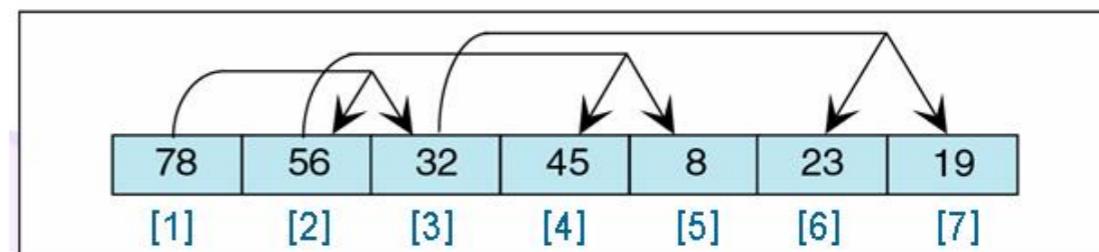
ตัวอย่าง **Heap** ที่ถูกต้อง



การแทน Heap ในหน่วยความจำ



(a) A heap in its logical form



(b) A heap in an array

Section 6

Huffman Code

บีบอัด file โดยไม่ให้เสียข้อมูลไปเลยได้อย่างไรให้ M เป็น ข้อมูลที่เราต้องการบีบอัด สมมติว่ามันมีขนาด 100,000 characters ซึ่งประกอบไปด้วยตัวอักษร a ถึง e เท่านั้นสมมติให้ 1 character ใช้ 1 byte (8 bits) ละนั้น 100,000 characters ใช้ 800,000 bits เราสามารถลดจำนวน bits ที่ใช้แต่ละตัวอักษรได้หรือไม่? เนื่องด้วยมีตัวอักษรเพียง 5 ตัว ดังนั้น สมมติให้ 1 character ใช้ 3 bits

‘a’	→	000
‘b’	→	001
‘c’	→	010
‘d’	→	011
‘e’	→	100

ดังนั้น จำนวน bits ที่ใช้จะลดลงเหลือ 300,000 bits ถ้าลดจำนวนบิตลงอีก

‘a’	→	0
‘b’	→	1
‘c’	→	00
‘d’	→	01
‘e’	→	10

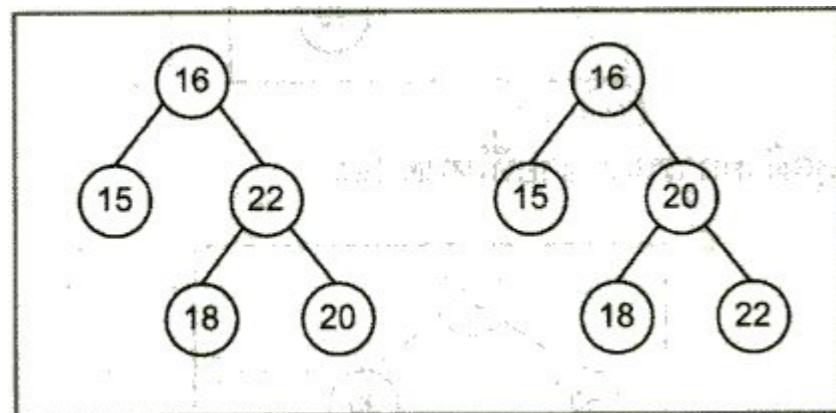
จะเห็นว่าจำนวน bits ที่ใช้จะลดลงแต่สมมติข้อมูล 001010 สามารถเป็นได้ทั้ง aababa หรือ cbda ก็ได้ !!

ใช้ Huffman Code เป็นรหัสแทนตัวอักษรที่แต่ละตัวอักษรมีความยาวของรหัสแตกต่างกันโดย ตัวอักษรที่ใช้บ่อย จะมีขนาดสั้นตัวอักษรที่ใช้น้อย จะมีขนาดยาวทั้งนี้เพื่อทำให้ข้อมูลมีขนาดเล็กลง

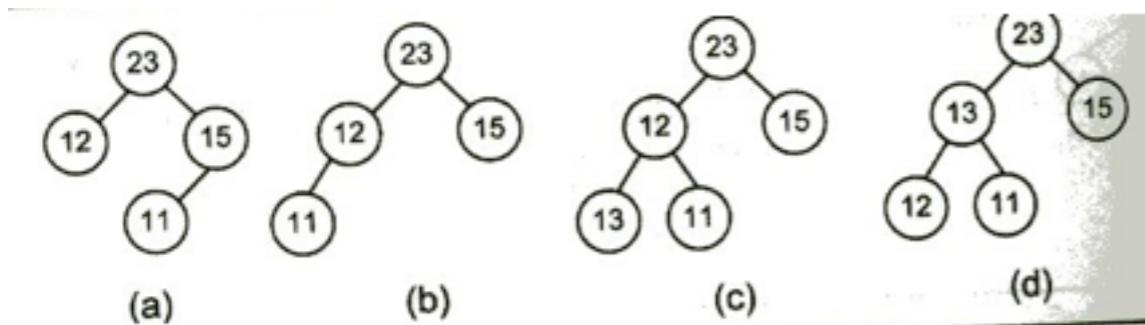
Section 7

แบบฝึกหัด

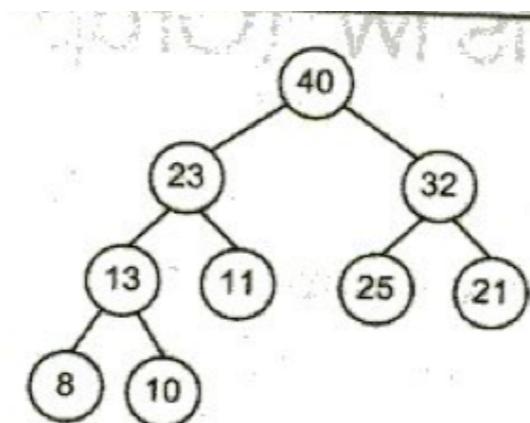
1. จากใบナรีเลิร์ชทรีต่อไปนี้ อยากร้าบว่ารูปใดเป็น BST ที่ถูกต้องและรูปใดเป็น BST ที่ไม่ถูกต้องจงอธิบาย



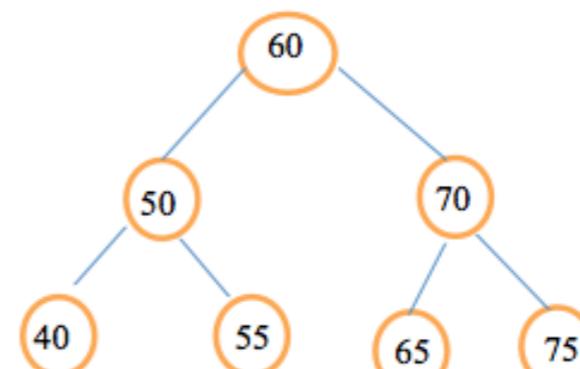
2. จากโครงสร้างทรีต่อไปนี้ อยากร้าบว่ารูปใดคืออีพ หรือรูปใดไม่ใช่ เพราะอะไรจงอธิบาย



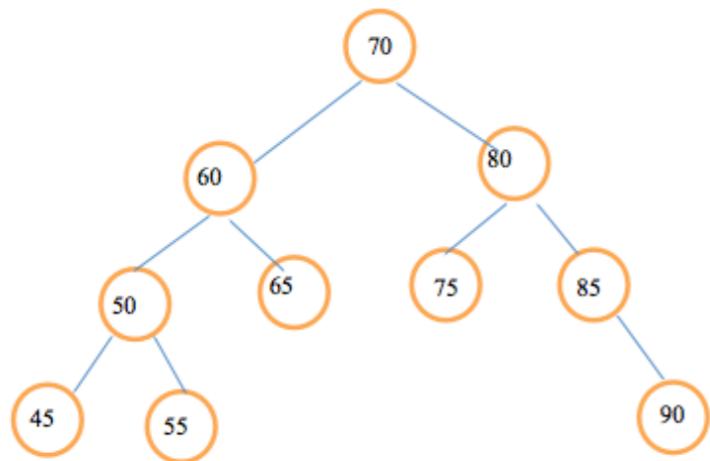
3. จงหาดอีพทรีต่อไปนี้ลงในโครงสร้างอาร์เรย์



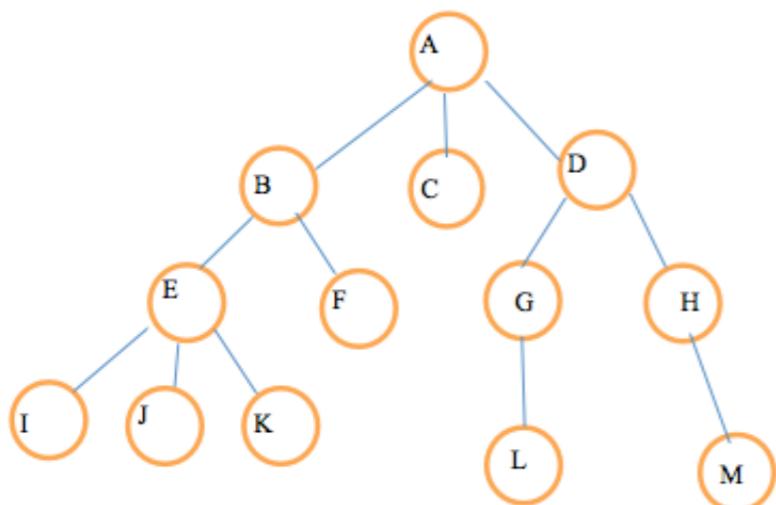
4. จงทำการแทรกโหนด 44 , 66, 77 ลงในใบナรีเลิร์ชทรีต่อไปนี้



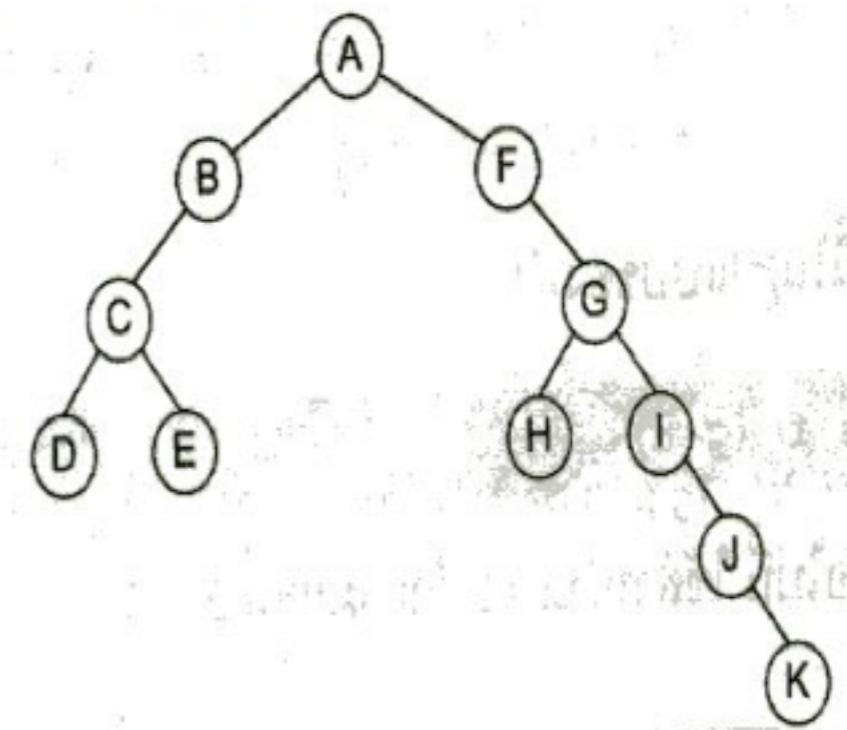
5. จากใบnaireเสิร์ชทรีต่อไปนี้ จงเขียนใบnaireเสิร์ชทรีใหม่ หลังจากที่ได้ดำเนินการลบข้อมูล 60 และโหนด 85 ออกจากทรี



6. จาก General Tree ต่อไปนี้ จงแสดงขั้นตอนการแปลงเป็นใบnaireทรี(Binary Tree)



7. จากทรีที่กำหนดให้ต่อไปนี้ จงทองทรีแบบ Dept-first



การเรียงลำดับข้อมูล Sorting



วัตถุประสงค์

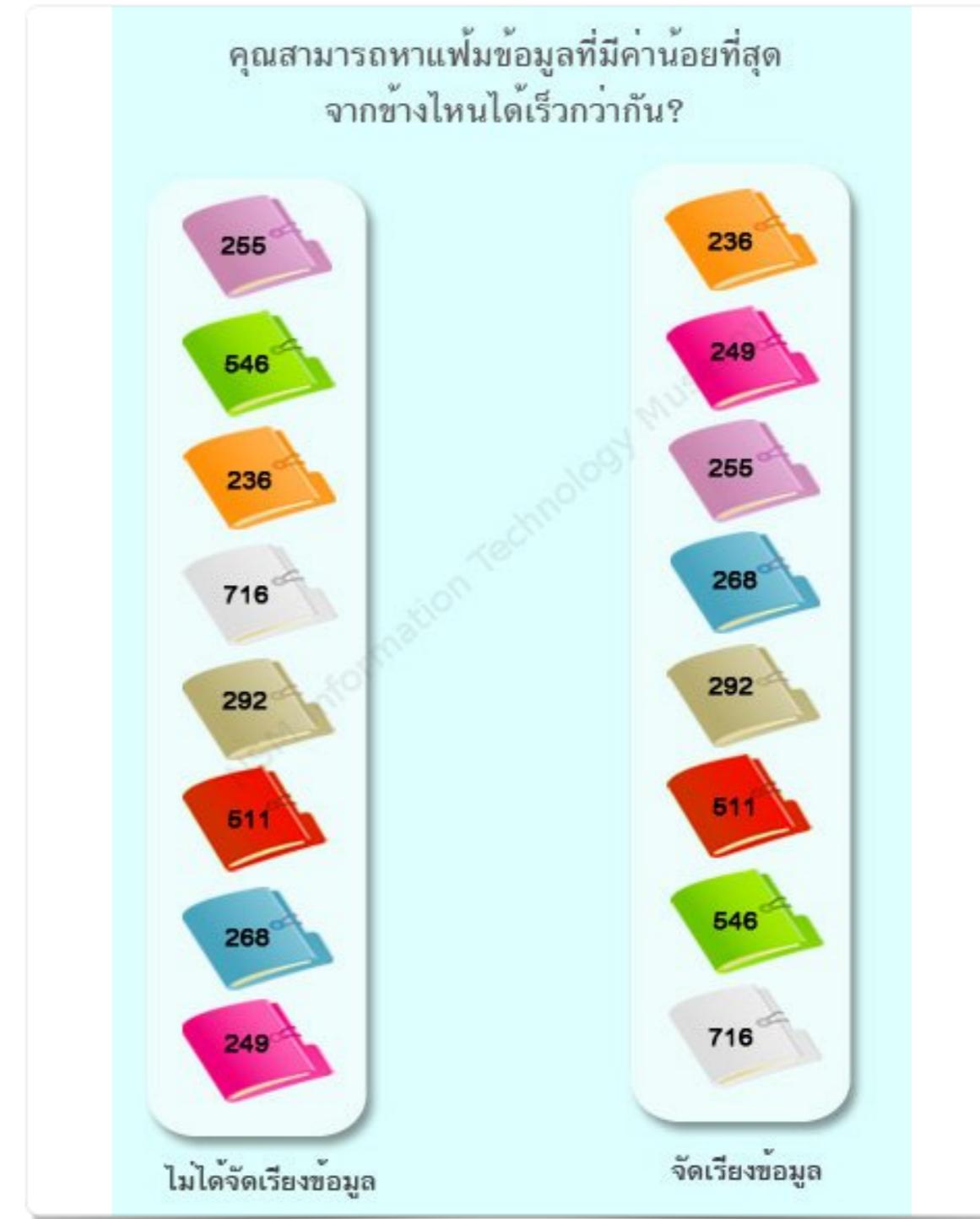
1. เห็นความสำคัญของการเรียงลำดับข้อมูลที่มักนำไปใช้ในการประมวลผล
2. บอกประเภทการเรียงลำดับข้อมูล พร้อมอธิบาย
3. เข้าใจการทำงานของแต่ละอัลกอริธึมที่ใช้เรียงลำดับข้อมูล

Section 1

การเรียงลำดับ (Sorting)

การเรียงลำดับข้อมูลเป็นการจัดการข้อมูลที่กระทำการมากในงานประยุกต์ต่างๆ มากมาย ยกตัวอย่างเช่น การนำข้อมูลนักศึกษามาจัดเรียงลำดับตามรหัสนักศึกษา เพื่อนำไปใช้ในการพิมพ์ใบเซ็นชื่อเข้าสอบ หรือการเรียงข้อมูลพนักงานตามรหัสของพนักงานเพื่อใช้ในการพิมพ์สลิปเงินเดือน เป็นต้น

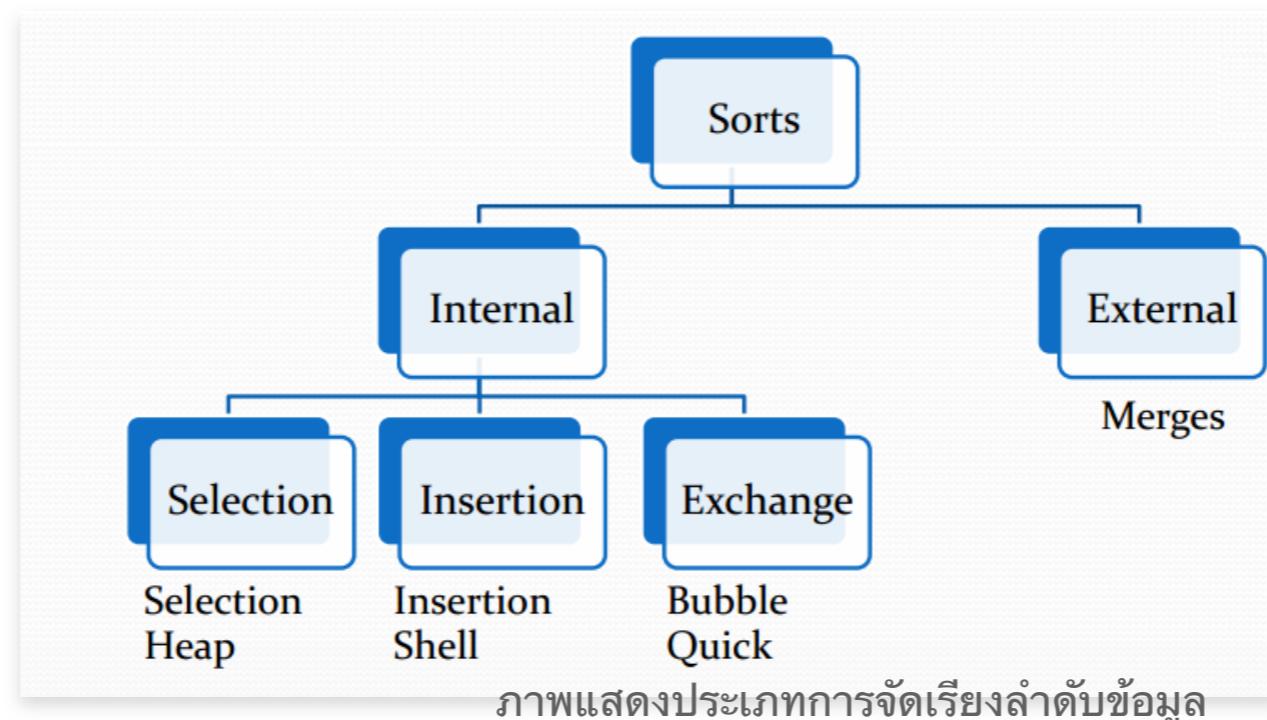
ถ้ากำหนดให้ A เป็นลิสต์ขนาด n อิลิเมนต์ มี $A_1, A_2, A_3, \dots, A_n$ เก็บอยู่ในหน่วยความจำ การจัดเรียงข้อมูลจะสามารถทำได้ $n!$ แบบ แต่โดยทั่วไปการจัดเรียงข้อมูลที่นิยมคือ การจัดเรียงลำดับข้อมูลจากน้อยไปมาก หรือจากมากไปน้อยเท่านั้น ซึ่งจะเลือกใช้อัลกอริทึมใดในการเรียงลำดับนั้น ขึ้นอยู่กับลักษณะของข้อมูลได้แก่ ขนาดข้อมูล และประสิทธิภาพการจัดเรียงของแต่ละอัลกอริทึมเมื่อเทียบกับปริมาณข้อมูล



ประเภทของการจัดเรียงลำดับข้อมูล การจัดเรียงลำดับข้อมูลในระบบคอมพิวเตอร์ สามารถแบ่งออกได้เป็น 2 ประเภทใหญ่ๆ

1. การจัดเรียงลำดับภายใน (Internal Sorting) เป็นการจัดเรียงลำดับข้อมูลที่เก็บอยู่ในหน่วยความจำ โดยข้อมูลเหล่านี้จะถูกเก็บอยู่ในโครงสร้างข้อมูลแบบอาร์เรย์ หรือลิงค์ลิสต์ ข้อมูลที่ทำการเรียงลำดับมีขนาดเล็กหรือจำนวนไม่มาก ซึ่งหน่วยความจำสามารถจะอ่านข้อมูลทั้งหมดขึ้นมาบนหน่วยความจำ และสามารถทำงานต่างๆ บนหน่วยความจำได้โดยไม่ต้องอาศัยสื่อบันทึกข้อมูล เช่น ดิสก์ หรือ เทป มาช่วยในการทำงาน ประสิทธิภาพของการจัดเรียงในลักษณะนี้ เน้นที่การสลับหรือเคลื่อนย้ายข้อมูลให้น้อยที่สุด จะทำให้ความเร็วของโปรแกรมดีขึ้น

2. การจัดเรียงลำดับภายนอก (External Sorting) เป็นการจัดเรียงลำดับข้อมูลที่เก็บอยู่ในสื่อบันทึกข้อมูล โดยทั่วไปข้อมูลที่บันทึกนี้ มักมีจำนวนมากจนไม่สามารถจะเก็บเอาไว้ในหน่วยความจำได้ทั้งหมด ต้องแบ่งออกเป็นส่วนย่อยๆ แล้วจึงนำมาจัดเรียงในหน่วยความจำ จากนั้นจึงทำการบันทึกกลับลงไปในสื่อสำหรับบันทึกข้อมูลเป็นส่วนๆ ต่อจากนั้นนำส่วนต่างๆ ที่จัดเรียงลำดับเรียบร้อยแล้วมาทำการรวมเข้าด้วยกัน (Merge) สำหรับการเรียงแบบภายนอกนั้น จะต้องคิดถึงเวลาที่สูญเสียไปอันเนื่องจากการถ่ายเทข้อมูลระหว่างเทปหรือดิสก์ กับหน่วยความจำหลักด้วย เวลาที่สูญเสียไปในการถ่ายเทข้อมูลระหว่างหน่วยความจำหลักกับเทป หรือดิสก์จะเป็นตัวระบุความดีเลວของแบบการคำนวณแบบเรียงภายนอก



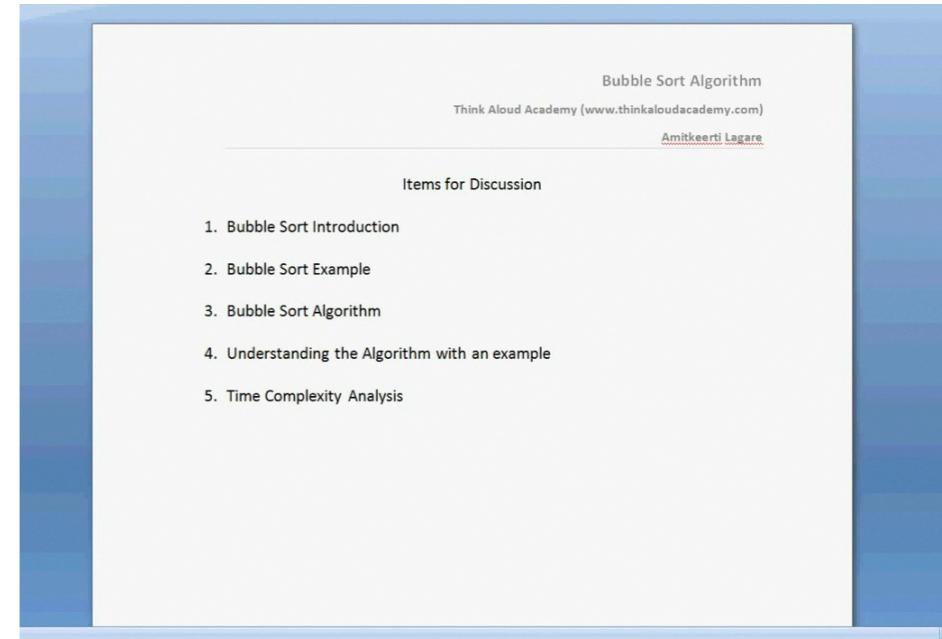
Section 2

การเรียงลำดับภายใน (Internal Sorting)

การจัดเรียงข้อมูลแบบ Bubble Sort

- ❖ ใช้การเปรียบเทียบคีย์ในตำแหน่งที่อยู่ติดกันทีละคู่
- ❖ ถ้าคีย์ที่เปรียบเทียบไม่อยู่ในตำแหน่งที่ต้องการแล้ว ให้ทำการสลับที่กันระหว่างข้อมูล 2 ตัวนั้น
- ❖ ทำเช่นนี้จนกระทั่งเปรียบเทียบครบทุกตัว ซึ่งคือ $N-1$ ครั้ง ทิศทางการทำงานอาจจะทำจากคูช้ายไปขวา หรือคูขวาไปช้าย หรือคูบันลงล่าง หรือคูล่างขึ้นบน ก็ได้
- ❖ ในแต่ละรอบของการเปรียบเทียบ คีย์ที่มีค่ามากจะถูกสลับที่ไปอยู่ในตำแหน่งตอนท้าย
- ❖ หรือคีย์ที่มีค่าน้อยจะถูกสลับไปอยู่ในตำแหน่งตอนบน สำหรับตัวอย่างนี้จะเริ่มเปรียบเทียบข้อมูลคู่ท้ายก่อน
- ❖ โดยให้ข้อมูลที่มีค่ามากสลับไปอยู่ด้านท้ายของข้อมูล หรือข้อมูลที่มีค่าข้อมูลมากอยู่ทางด้านล่างของแท่นั่นเอง

Movie 8.1 การจัดเรียงข้อมูลด้วย Bubble Sort



ที่มา : <http://www.youtube.com/watch?v=lyZQPjUT5B4>

ตัวอย่างแสดงการจัดเรียงข้อมูลด้วย Bubble Sort จาก

ข้อมูล 36 27 23 33 32 18 5 13 9 10

Unsorted	รอบที่ 1 เปรียบเทียบทั้งหมด 9 ครั้ง	รอบที่ 2 เปรียบเทียบทั้งหมด 8 ครั้ง
36	36 36 36 36 36 36 36 36 5	5 5 5 5 5 5 5 5 5
27	27 27 27 27 27 27 27 5 36	36 36 36 36 36 36 36 9
23	23 23 23 23 23 23 5 27 27	27 27 27 27 27 27 9 36
33	33 33 33 33 33 5 23 23 23	23 23 23 23 23 9 27 27
32	32 32 32 32 5 33 33 33 33	33 33 33 33 9 23 23 23
18	18 18 18 5 32 32 32 32 32	32 32 32 9 33 33 33 33
5	5 5 18 18 18 18 18 18 18	18 18 9 32 32 32 32 32
13	13 9 9 9 9 9 9 9 9	9 18 18 18 18 18 18 18
9	9 13 13 13 13 13 13 13 13	10 10 10 10 10 10 10 10
10	10 10 10 10 10 10 10 10 10	13 13 13 13 13 13 13 13

รอบที่ 3 เปรียบเทียบ 7 ครั้ง	รอบที่ 4 เปรียบเทียบ 6 ครั้ง	รอบที่ 5 = 5 ครั้ง
5 5 5 5 5 5 5 5	5 5 5 5 5 5 5	5 5 5 5 5 5 5
9 9 9 9 9 9 9 9	9 9 9 9 9 9 9	9 9 9 9 9 9 9
36 36 36 36 36 36 36 10	10 10 10 10 10 10 10	10 10 10 10 10 10
27 27 27 27 27 27 10 36	36 36 36 36 36 13	13 13 13 13 13 13
23 23 23 23 23 10 27 27	27 27 27 27 13 36	36 36 36 36 18
33 33 33 10 23 23 23 23	23 23 23 13 27 27	27 27 27 18 36
32 32 32 10 33 33 33 33	33 33 13 23 23 23	23 23 18 27 27
18 18 10 32 32 32 32 32	32 13 33 33 33 33	33 18 23 23 23 23
10 10 18 18 18 18 18 18	13 32 32 32 32 32	18 33 33 33 33 33
13 13 13 13 13 13 13 13	18 18 18 18 18 18	32 32 32 32 32 32

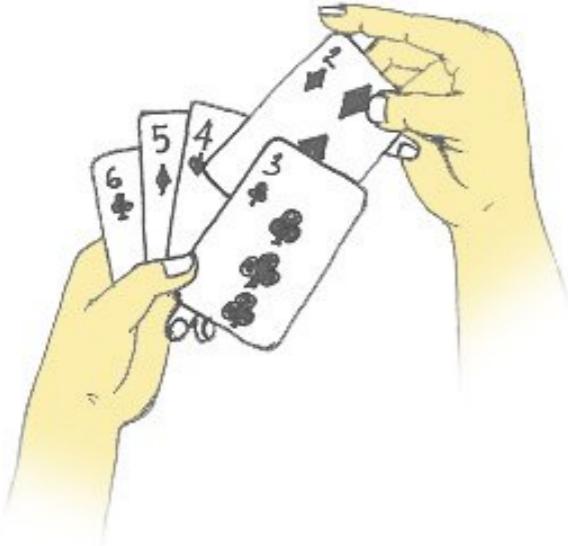
รอบที่ 6 = 4 ครั้ง	รอบที่ 7 = 3	8 = 2	9 = 1
5 5 5 5 5	5 5 5	5 5	5
9 9 9 9 9	9 9 9	9 9	9
10 10 10 10 10	10 10 10	10 10	10
13 13 13 13 13	13 13 13	13 13	13
18 18 18 18 18	18 18 18	18 18	18
36 36 36 36 23	23 23 23	23 23	23
27 27 27 23 36	36 27 27	27 27	27
23 23 23 27 36	27 36 36	36 36	32
33 32 32 32 32	32 32 32	32 32	36
32 33 33 33 33	33 33 33	33 33	33

เมื่อพิจารณาวิธีการจัดเรียงแบบ Bubble Sort กรณีที่ **แก้ไขที่สุด** ในการจัดเรียงจะเป็นในกรณีที่มีการสลับตำแหน่งและการเปรียบเทียบในทุกๆ ครั้งที่มีการเปรียบเทียบ

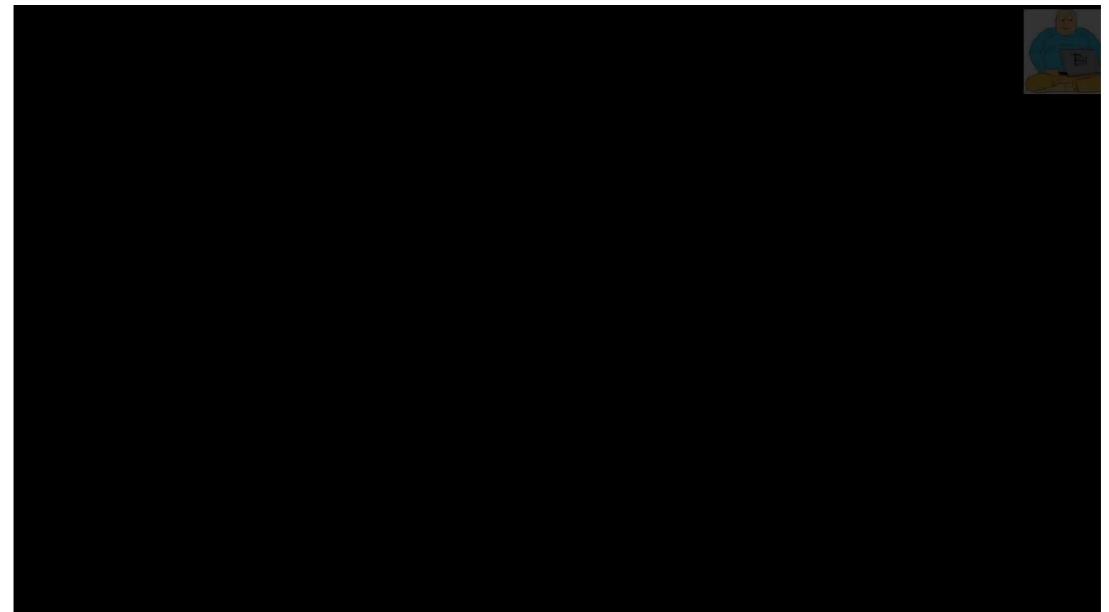
การจัดเรียงลำดับข้อมูลแบบ Insertion Sort

เทคนิคมาจากการลักษณะการจัดไฟร์ในมือของผู้เล่น คือ เมื่อผู้เล่นได้ไฟร์ใบใหม่ เพิ่มขึ้นมา จะนำไฟร์ใบนั้นไปแทรกในตำแหน่งที่เหมาะสม ทำให้ไฟร์ในมือบางส่วนต้องขยับตำแหน่งออกไป ซึ่งการจัดเรียงลำดับข้อมูลแบบแทรกนี้ จะเริ่มพิจารณาคีย์ในตำแหน่งที่ 2 เป็นต้นไป โดยนำคีย์ที่พิจารณาไปแทรกในตำแหน่งที่ถูกต้อง และจะมีผลให้คีย์ในตำแหน่งที่อยู่หลังตำแหน่งที่แทรกขยับตำแหน่งออกไปเรื่อยๆ

จากวิธีการดังกล่าว ถ้าการพิจารณาคีย์มาถึงตำแหน่งที่ 1 จะเห็นได้ว่าข้อมูลจะถูกแบ่งออกเป็นสองส่วน คือ ส่วนแรกจะเป็นส่วนที่ถูกจัดเรียงลำดับตามคีย์แล้ว อีกส่วนจะเป็นส่วนที่ยังไม่ได้ทำการจัดเรียง



Movie 8.2 แสดงการจัดเรียงข้อมูลแบบ Insertion Sort



ที่มา : <http://www.youtube.com/watch?v=DFG-XuyPYUQ>

ตัวอย่างแสดงการจัดเรียงข้อมูลด้วย Insertion Sort

ข้อมูลเดิม	44	33	11	55	77	90	40	60	99	22	88	66
$i = 1$	44	33	11	55	77	90	40	60	99	22	88	66
$i = 2$	11	33	44	55	77	90	40	60	99	22	88	66
$i = 3$	11	22	44	55	77	90	40	60	99	33	88	66
$i = 4$	11	22	33	55	77	90	40	60	99	44	88	66
$i = 5$	11	22	33	40	77	90	50	60	99	44	88	66
$i = 6$	11	22	33	40	44	90	55	60	99	77	88	66
$i = 7$	11	22	33	40	44	55	90	60	99	77	88	66
$i = 8$	11	22	33	40	44	55	60	90	99	77	88	66
$i = 9$	11	22	33	40	44	55	60	66	99	77	88	90
$i = 10$	11	22	33	40	44	55	60	66	77	99	88	90
$i = 11$	11	22	33	40	44	55	60	66	77	88	99	90
$i = 12$	11	22	33	40	44	55	60	66	77	88	90	99

การจัดเรียงลำดับแบบแทรก จะมีการจัดเรียงลำดับข้อมูลทั้งหมด $n - 1$ รอบ แต่มักใช้เวลาอ่อนกว่าการจัดเรียงแบบ Bubble ซึ่งในการจัดเรียงแต่ละรอบนั้น จำนวนการเปรียบเทียบคีย์จะไม่แน่นอน เพราะในแต่ละรอบการเปรียบเทียบจะสิ้นสุดเมื่อไม่มีการสลับตำแหน่งของคีย์ ดังนั้น จะพิจารณาจำนวนการเปรียบเทียบคีย์เป็น 3 กรณีคือ

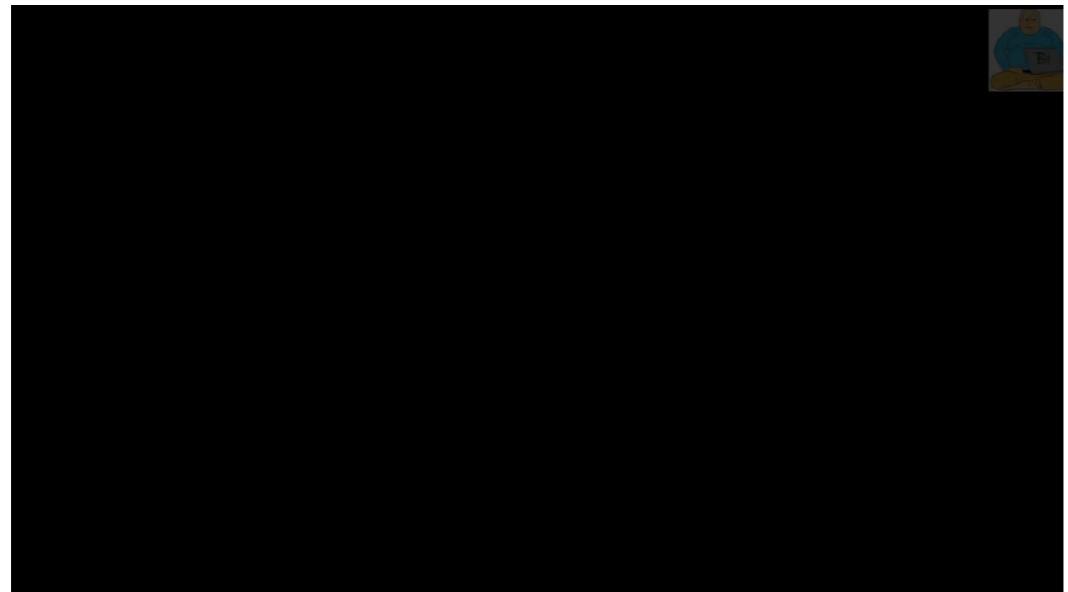
กรณีดีที่สุด ข้อมูลถูกจัดเรียงลำดับเรียบร้อยแล้ว กรณีนี้แต่ละรอบจะมีการเปรียบเทียบคีย์เพียงครั้งเดียว เพราะฉะนั้นจำนวนการเปรียบเทียบคีย์คือ $n - 1$

กรณีแย่ที่สุด ข้อมูลถูกจัดเรียงลำดับกลับกัน คือ เรียงลำดับค่าคีย์จากมากไปน้อย (ในกรณีที่ต้องการจัดเรียงลำดับจากน้อยไปมาก)

การจัดเรียงข้อมูลแบบ Selection Sort

- ❖ เป็นวิธีที่ง่ายที่สุด โดยเริ่มจาก เลือกค่าของข้อมูลที่มีค่าน้อยที่สุด
- ❖ นำมาแลกเปลี่ยนกับค่าในตำแหน่งแรกสุดของกลุ่ม
- ❖ หลังจากนั้นก็ทำการทำตามหลักการทั้ง 2 กับข้อมูลที่เหลือ คือในครั้งที่ 2 ค่า A(2) จะถูกแลกเปลี่ยนกับค่าที่เลือกแล้วว่ามีค่าน้อยที่สุด ในลิสต์ A(2)...A(N)
- ❖ และในครั้งที่ 3 ค่า A(3) จะถูกแลกเปลี่ยนกับค่าที่เลือกแล้วว่ามีค่าน้อยที่สุด ในลิสต์ A(3)...A(N)
- ❖ และเรื่อยไปจนกระทั่งเหลือข้อมูลที่จะเปรียบเทียบแค่ 2 ค่าคือ A(N-1) และ A(N) ดังนั้น จำนวนรอบในการกระทำเป็น $n-1$ รอบ

Movie 8.3 แสดงการจัดเรียงข้อมูลด้วย Selection Sort



ที่มา : <http://www.youtube.com/watch?v=EdUWYka7kpI>

ตัวอย่าง แสดงการจัดเรียงข้อมูลด้วย Selection Sort

ข้อมูลเดิม	44	33	11	55	77	90	40	60	99	22	88	66
I = 1	44	33	11	55	77	90	40	60	99	22	88	66
I = 2	11	33	44	55	77	90	40	60	99	22	88	66
I = 3	11	22	44	55	77	90	40	60	99	33	88	66
I = 4	11	22	33	55	77	90	40	60	99	44	88	66
I = 5	11	22	33	40	77	90	50	60	99	44	88	66
I = 6	11	22	33	40	44	90	55	60	99	77	88	66
I = 7	11	22	33	40	44	55	90	60	99	77	88	66
I = 8	11	22	33	40	44	55	60	90	99	77	88	66
I = 9	11	22	33	40	44	55	60	66	99	77	88	90
I = 10	11	22	33	40	44	55	60	66	77	99	88	90
I = 11	11	22	33	40	44	55	60	66	77	88	99	90
I = 12	11	22	33	40	44	55	60	66	77	88	90	99

การจัดเรียงลำดับข้อมูลแบบ Quick Sort

- ❖ มีแนวความคิดในการจัดเรียงลำดับคือ จะเลือกคีย์จากกลุ่มข้อมูลขึ้นมา 1 ค่า
- ❖ และใช้คีย์นั้นเป็นหลักในการแบ่งข้อมูลเป็นสองส่วน ซึ่งผลของการแบ่งจะได้
- ❖ ส่วนหนึ่งอยู่ในตำแหน่งต่อนหน้าและคีย์ที่อยู่ในส่วนนี้จะมีค่าน้อยกว่าคีย์ที่เลือก ส่วนอีกส่วนหนึ่งจะอยู่ในตำแหน่งต่อหลัง
- ❖ และคีย์ในส่วนนี้จะมีมากกว่าหรือเท่ากับคีย์ที่เลือก
- ❖ เมื่อแบ่งสำเร็จแล้ว ให้นำแต่ละส่วนไปแบ่งเป็นส่วนย่อย ในแบบเดียวกันต่อไป
- ❖ และให้ดำเนินการแบ่งส่วนย่อยต่อไปในลักษณะเดียวกัน จนกระทั่งทุกส่วนไม่สามารถแบ่งออกเป็นส่วนย่อยต่อไปได้อีก

Movie 8.4 แสดงการเรียงลำดับข้อมูลแบบ Quick Sort



ที่มา : <http://www.youtube.com/watch?v=2DK8qMHhqkE>

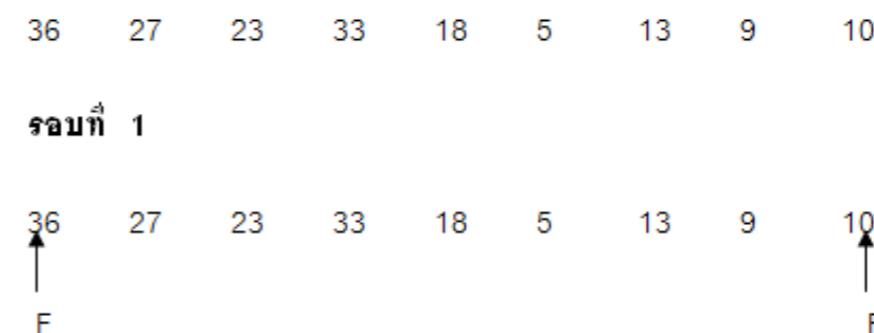
สำหรับวิธีการแบ่งส่วนเพื่อให้มีคุณสมบัติดังกล่าวมีด้วยกันหลายแบบ เช่น เลือกคีย์ที่อยู่ในตำแหน่งแรก หรือค่ากลาง (medium) ของจำนวนข้อมูล แต่ในที่นี้จะใช้วิธีแรกคือ เลือกคีย์ที่อยู่ในตำแหน่งแรกของส่วนที่จะแบ่ง สมมติว่า X จากนั้นให้พิจารณาคีย์จากตำแหน่งตอนท้ายขึ้นมาข้างหน้าจนกว่าจะพบคีย์ที่มีค่าน้อยกว่า X และให้สลับในตำแหน่งนั้นกับ X ต่อไปให้พิจารณาคีย์จากตำแหน่งตอนหน้าไปข้างหลัง โดยเริ่มลดจากตำแหน่งที่เกิดการสลับจนกว่าจะพบคีย์ที่มีค่ามากกว่าหรือเท่ากับ X และให้สลับคีย์ในตำแหน่งนั้นกับ X ซึ่งอยู่ในตำแหน่งใหม่แล้วจากนั้นไปพิจารณาคีย์จากตำแหน่งตอนท้ายขึ้นมาข้างหน้าอีก โดยเริ่มลดจากตำแหน่งที่เกิดการสลับ และจะดำเนินการในลักษณะเดียวกันสลับกันไป จนการพิจารณาครบกับตำแหน่งของ X ในขณะนั้น

การแบ่งส่วนด้วยวิธีดังกล่าว คีย์ถูกหน่วยจะถูกสลับตำแหน่งไปมา และเมื่อสิ้นกระบวนการคีย์ที่ถูกเลือกจะไปต่อกันในตำแหน่งที่คีย์นี้ต้องอยู่พอดี ดังตัวอย่างต่อไปนี้

การเรียงข้อมูลจะเริ่มโดยใช้ตัวชี้ 2 ตัว คือ F และ R โดยในครั้งแรกให้ F ชี้ไปที่ 1 ซึ่งก็คือคีย์ตัวแรก และ R ชี้ไปยังคีย์ตัวสุดท้ายในลิสต์

การเปรียบเทียบจะกระทำระหว่างค่าที่ถูกชี้โดย F และ R ดังตัวอย่าง

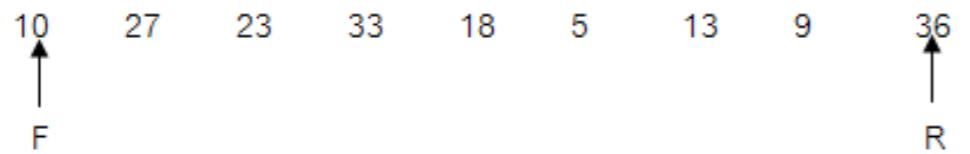
สมมติ A ประกอบด้วยตัวเลข 9 จำนวน และคีย์ที่ถูกเลือก คือ คีย์ตัวแรก 36



เริ่มเปรียบเทียบจากขวาไปซ้าย โดยเริ่มจากค่าที่ R ซึ่งอยู่ เพื่อหาตัวเลขที่มีค่าน้อยกว่าค่าคีย์หรือ ตัวชี้ F และ R ชี้ไปที่คีย์ที่เลือกไว้ ซึ่งตอนนี้ ค่าที่ R ซึ่งอยู่ มีค่าน้อยกว่าค่าของคีย์ ($10 < 36$) จึงทำการสลับตำแหน่งระหว่างค่าคีย์กับค่าที่ R ซึ่งอยู่ ซึ่งจะได้ลิสต์เป็น



หลังจากนั้นเริ่มเปรียบเทียบจากซ้ายไปขวา โดยเริ่มจากค่าที่ F ซึ่งอยู่ และเพิ่มค่า F ขึ้นครั้งละ 1 โดยการเปรียบเทียบจะหยุดเมื่อพบคีย์ที่มีค่ามากกว่า 36 หรือ ตัวชี้ F และ R ชี้ไปที่คีย์ที่เลือกไว้



เปรียบเทียบ ค่าที่ F ชี้อยู่ กับค่าคีร์ ซึ่ง 10 มีค่าน้อยกว่า 36 จึงเพิ่มค่า F ขึ้นหนึ่ง



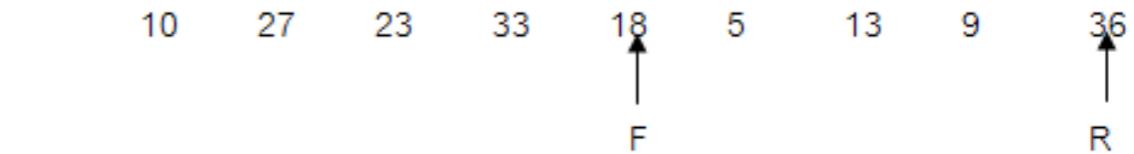
เปรียบเทียบ ค่าที่ F ชี้อยู่ กับค่าคีร์ ซึ่ง 27 มีค่าน้อยกว่า 36 จึงเพิ่มค่า F ขึ้นหนึ่ง



เปรียบเทียบ ค่าที่ F ชี้อยู่ กับค่าคีร์ ซึ่ง 23 มีค่าน้อยกว่า 36 จึงเพิ่มค่า F ขึ้นหนึ่ง



เปรียบเทียบ ค่าที่ F ชี้อยู่ กับค่าคีร์ ซึ่ง 33 มีค่าน้อยกว่า 36 จึงเพิ่มค่า F ขึ้นหนึ่ง



เปรียบเทียบ ค่าที่ F ชี้อยู่ กับค่าคีร์ ซึ่ง 18 มีค่าน้อยกว่า 36 จึงเพิ่มค่า F ขึ้นหนึ่ง



เปรียบเทียบ ค่าที่ F ชี้อยู่ กับค่าคีร์ ซึ่ง 5 มีค่าน้อยกว่า 36 จึงเพิ่มค่า F ขึ้นหนึ่ง



เปรียบเทียบ ค่าที่ F ชี้อยู่ กับค่าคีร์ ซึ่ง 13 มีค่าน้อยกว่า 36 จึงเพิ่มค่า F ขึ้นหนึ่ง



เปรียบเทียบ ค่าที่ F ชี้อยู่ กับค่าคีร์ ซึ่ง 9 มีค่าน้อยกว่า 36 จึงเพิ่มค่า F ขึ้นหนึ่ง

10 27 23 33 18 5 13 9 

ซึ่งในที่นี้ไม่พบค่าที่มากกว่าคีย์ และทั้งตัวชี้ F และ R ชี้ไปที่คีย์ หมายถึงว่า ตอนนี้ข้อมูลทุกตัวได้ถูกเปรียบเทียบ กับ 36 แล้ว ณ จุดนี้ คีย์ 36 จะอยู่ในตำแหน่งที่ถูกต้อง และได้แบ่งลิสต์ออกเป็น 2 ลิสต์ย่อย คือ

(10 27 23 33 18 5 13 9) กับ (36)

การเรียงในรอบต่อไปจะทำกับข้อมูลในแต่ละกลุ่ม การกระทำขั้นตอนซ้ำในแต่ละกลุ่มย่อยจะกระทำเมื่อมีข้อมูลในกลุ่มย่อยตั้งแต่ 2 ค่าขึ้นไป เนื่องจากเราจะทำได้ครั้งละกลุ่ม เท่านั้น จึงจำเป็นต้องมีการเก็บตำแหน่งเริ่มต้น กับตำแหน่งสุดท้ายของลิสต์ที่ยังไม่ได้ถูกเรียงเอาไว้ ซึ่งจะต้องใช้ สแตกช่วย

แต่เนื่องจาก sublist ย่อย (36) มีค่าเดียวจึงไม่ต้องนำมาเรียง ดังนั้นจึงไม่ต้องเก็บตำแหน่งเริ่มต้นหรือตำแหน่งสุดท้ายไว้ในสแตก ซึ่งในกรณีแสดงว่าค่า 36 อยู่ในตำแหน่งที่ถูกต้องแล้ว

รอบที่ 2

ในรอบนี้จะต้องทำการเรียงข้อมูลใน sublist แรก โดยให้ F ชี้ไปที่ตำแหน่งแรกของ sublist และ R ชี้ไปที่ตำแหน่งสุดท้ายของ sublist และปฏิบัติเหมือนในรอบแรก

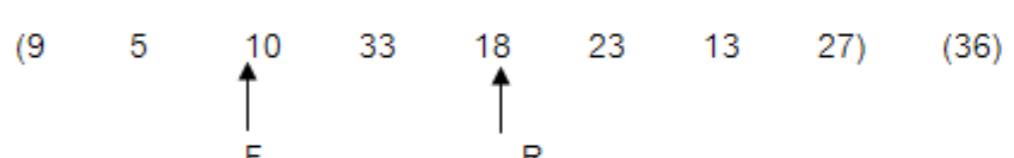




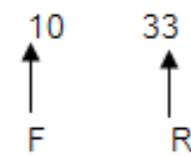




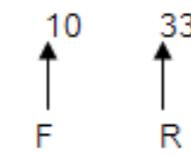




(9 5 10 33 18 23 13 27) (36)



(9 5 10 33 18 23 13 27) (36)



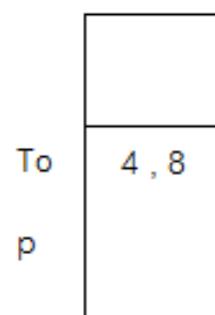
(9 5 10 33 18 23 13 27) (36)



จะได้ Sublist ย่อย คือ

(9 5) (10) (33 18 23 13 27) (36)

เนื่องจากมี 2 sublist ย่อย จะนำตำแหน่งเริ่มต้นกับตำแหน่งสุดท้ายของ sublist ชุดหลังเก็บไว้ในสแตกดังภาพ



รอบที่ 3

(9 5) (10) (33 18 23 13 27) (36)

(5 9) (10) (33 18 23 13 27) (36)

จะได้ Sublist ย่อย คือ

(5) (9) (10) (33 18 23 13 27) (36)

รอบที่ 4

ในรอบนี้จะต้องทำการ POP ตำแหน่งเริ่มต้นและตำแหน่งสุดท้ายของ sublist ที่ยังไม่ได้ทำการเรียง ได้ค่า 5 และ 8 คือต้องทำการเรียงข้อมูลในตำแหน่งที่ 5 ถึงตำแหน่งที่ 8

(5) (9) (10) (33 18 23 13 27) (36)



(5) (9) (10) (27) 18 23 13 33) (36)



(5) (9) (10) (27) 18 23 13 33) (36)



(5) (9) (10) (27) 18 23 13 33) (36)



(5) (9) (10) (27) 18 23 13 33) (36)



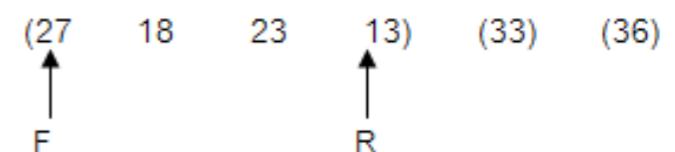
จะได้ Sublist ย่อย คือ

sublist 33 มีค่าเดียว แสดงว่าอยู่ในลำดับที่ถูกต้อง

แล้ว

รอบที่ 5

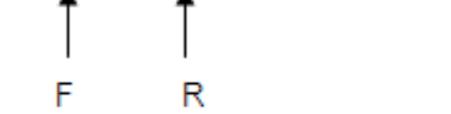
(5) (9) (10) (27) 18 23 13 33) (36)



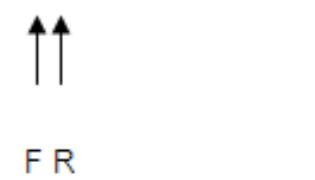
(5) (9) (10) (13) 18 23 27) (33) (36)



(5) (9) (10) (13) 18 23 27) (33) (36)



(5) (9) (10) (13) 18 23 27) (33) (36)



(5) (9) (10) (13) 18 23 27) (33) (36)



(5) (9) (10) (27) 18 23 13) (33) (36)



จะได้ Sublist ย่อย คือ

(5) (9) (10) (13 18 23) (27) (33) (36)

รอบที่ 6

(5) (9) (10) (13 ↑ 18 ↑ 23) (27) (33) (36)
F R

(5) (9) (10) (13 ↑ 18 ↑ 23) (27) (33) (36)
F R

(5) (9) (10) (13 ↑ ↑ 18 23) (27) (33) (36)
F R

จะได้ Sublist ย่อย คือ

(5) (9) (10) (13) (18 23) (27) (33) (36)

4

รอบที่ 7

(5) (9) (10) (13) (18) (23) (27) (33) (36)

↑
F R

(5) (9) (10) (13) (18) (23) (27) (33) (36)
↑
F R

จะได้ผลลัพธ์เป็น

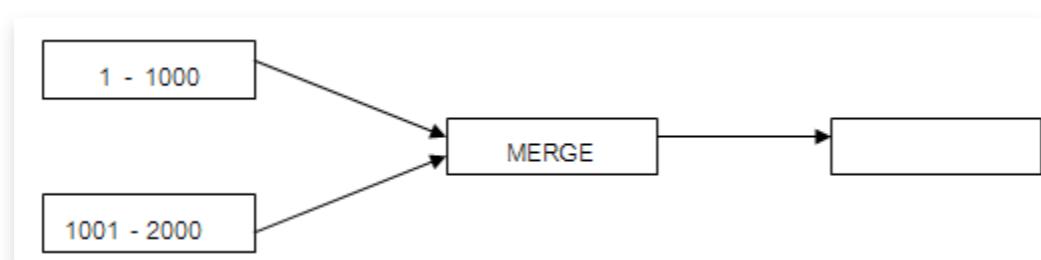
(5) (9) (10) (13) (18) (23) (27) (33) (36)

Section 3

การเรียงลำดับภายนอก (External Sorting)

การเรียงลำดับแบบภายนอกที่ได้ก่อตัวในตอนที่แล้วนั้น ข้อมูลที่จะเรียงจะต้องอยู่พร้อมกันในหน่วยความจำ ดังนั้นจึงไม่เหมาะสมสำหรับการเรียงข้อมูลที่มีจำนวนมาก ในหัวข้อนี้จะกล่าวถึงการเรียงข้อมูลจำนวนมาก ที่เราเรียกว่า external sorting

Sublist 1



ภาพแสดงการรวมไฟล์โดยใช้ Merge Sort
Sublist 2

ถ้าแฟ้มที่ต้องการเรียงลำดับมีขนาด 2000 รายการ สมมุติว่าหน่วยความจำสามารถจุข้อมูลได้เพียงครั้งละ 1000 รายการ วิธีแก้ปัญหาวิธีหนึ่งก็คือ ทำการเรียงลำดับแบบภายนอก 2 ครั้ง สำหรับรายการที่ 1 - 1000 ครั้งหนึ่ง และรายการที่ 1001 - 2000 อีกครั้งหนึ่ง จากนั้นจึงนำแฟ้มย่อยที่เรียงแล้วทั้ง 2 น้ำมารวบ (merge) กัน (แฟ้มย่อยที่ผ่านการเรียงลำดับแล้ว เรียกว่า รัน)

เทคนิคการเรียงลำดับแบบภายนอกเกือบทุกวิธี ใช้หลักการดังนี้คือ แบ่งรายการที่จะเรียงลำดับออกเป็นกลุ่มย่อย และจัดการเรียงลำดับแบบภายนอกกับรายการกลุ่มเหล่านั้น โดยการเก็บรายการกลุ่มย่อยเหล่านี้ ในสภาพของแฟ้ม ลำดับ จากนั้นจึงนำแฟ้มย่อยเหล่านี้มาทำการผสานเป็นแฟ้มเดียวกัน

เทคนิคของการเรียงลำดับแบบภายนอกอาจแตกต่างกัน ขึ้นอยู่กับ

- ❖ วิธีการเรียงลำดับแบบภายนอกที่เลือกใช้
 - ❖ การจัดสรรเนื้อที่หน่วยความจำสำหรับการเรียงลำดับแบบภายนอก
 - ❖ การกระจายของ run ต่าง ๆ ในอุปกรณ์บันทึกข้อมูล
 - ❖ จำนวน run ที่ใช้ในการผสานแต่ละรอบ
- การประเมินผลการทำงานของการเรียงลำดับแบบภายนอกนั้น วัดจากจำนวนครั้งของการเปรียบเทียบ การสลับค่าของข้อมูล N ตัว สำหรับการเรียงลำดับแบบภายนอกนั้น

ใช้วิธีดูเซ็นนี้ไม่ได้ เพราะความเร็วของ CPU กับ I/O ต่างกันมาก ดังนั้นการเรียงลำดับแบบนอกจึงวัดจากจำนวนครั้งในการถ่ายโอนข้อมูล ระหว่างหน่วยความจำกับอุปกรณ์บันทึกข้อมูลภายนอก โดยนับจากจำนวนรอบ (pass) ของการผสานข้อมูล นั่นคือ จำนวนครั้งที่แต่ละรายการต้องเกี่ยวข้องกับการผสานข้อมูล

ในการเรียงลำดับภายนอกนั้น เน้นความสำคัญของวิธีการผสานมากกว่าวิธีการเรียงลำดับแบบภายใน

การผสานของการเรียงลำดับภายนอก มีหลายวิธีดังนี้

- ❖ Two-way-merge
- ❖ The balanced merge
- ❖ The polyphase merge
- ❖ The cascade merge

การดำเนินการเรียงลำดับแบบภายนอก มี 2 ขั้นตอนดังนี้

1.) Phase Sort

แบ่งชุดข้อมูลออกเป็นกลุ่มย่อย ๆ (Sublist)

ดำเนินการจัดเรียงข้อมูลในแต่ละ Sublist ด้วยวิธีการแบบ Internal Sorting

จัดให้ Sublist ต่าง ๆ ที่เรียงลำดับแล้วอยู่ใน 2 Sublist (แฟ้มข้อมูลย่อย)

2.) Phase Merge เชื่อม 2 Sublist เข้าด้วยกันเป็น Sublist ใหม่ ด้วยวิธีการผสาน (Merging)

ในที่นี้จะศึกษาวิธีการแรกคือ Two-way merge Sort ก่อนที่จะศึกษาขั้นตอนวิธีการเรียงแบบนี้ ให้ทำความเข้าใจกับการ Merge ก่อน

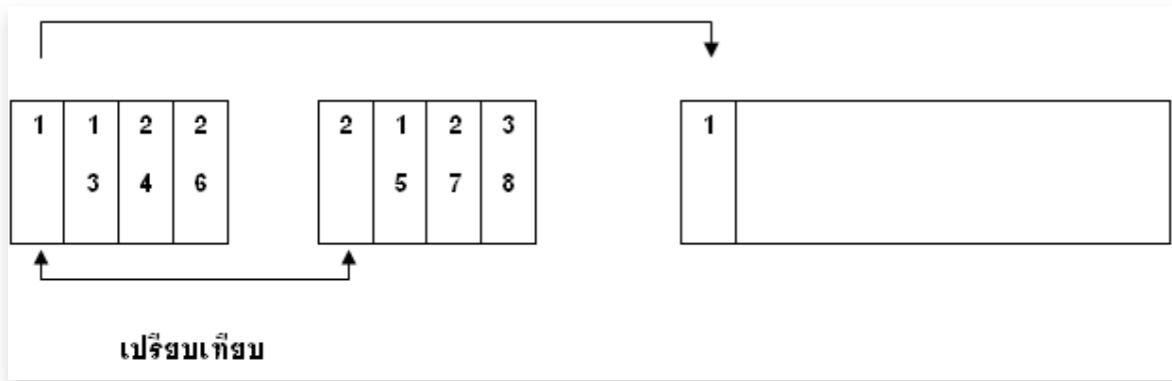
การ Merge

จะใช้ในกรณีที่ต้องการเรียงข้อมูลที่มีปริมาณมากๆ โดยจะแบ่งข้อมูลออกเป็น 2 กลุ่ม ซึ่งกลุ่มของข้อมูลที่แบ่งออกมานี้ ต้องไม่มากเกินขนาดของหน่วยความจำที่มีอยู่และนำข้อมูลแต่ละกลุ่มมาจัดเรียง ในหน่วยความจำโดยเลือกวิธีที่เหมาะสม อาจเป็น Selection sort หรือ Quick sort ข้อมูลที่จัดเรียงใหม่จะนำไปบันทึกในลีกภายนอก ซึ่งจะทำให้เราได้กลุ่มข้อมูล 2 กลุ่มที่ได้จัดเรียงลำดับข้อมูลแล้ว ดังตัวอย่าง สมมติว่าข้อมูลสองกลุ่มประกอบด้วย

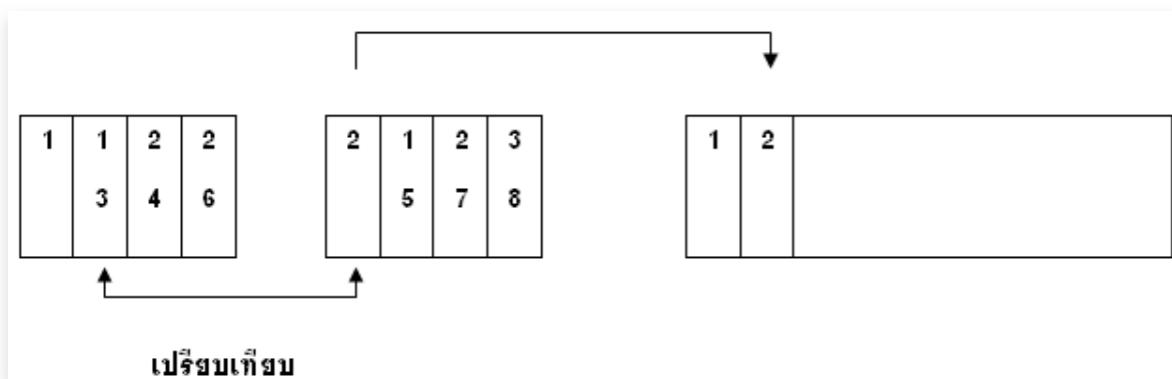
1	13	24	26
กลุ่มที่ 1			

2	15	27	38
กลุ่มที่ 2			

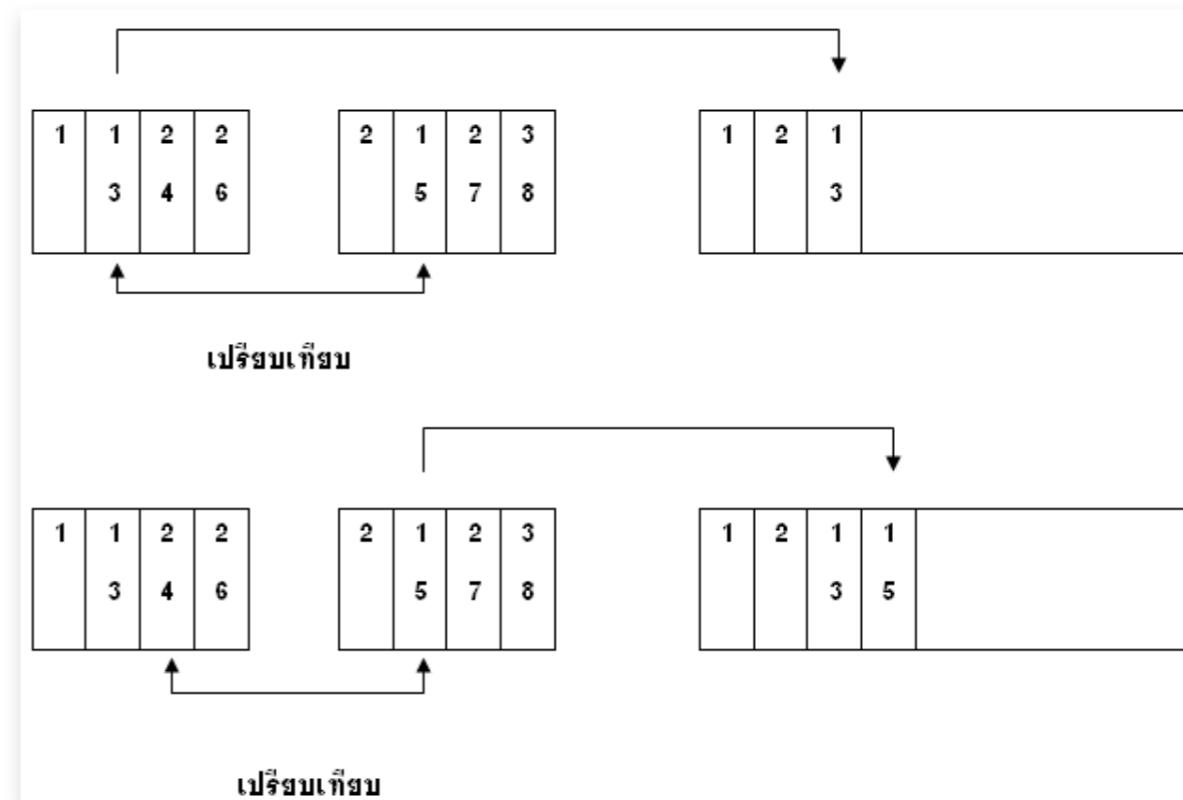
การเรียงลำดับข้อมูล 2 กลุ่มนี้เริ่มจากการเปรียบเทียบข้อมูลที่ละคู่ของข้อมูลทั้งสองกลุ่ม ในที่นี้คือค่า 1 ในกลุ่มที่ 1 และค่า 2 ในกลุ่มที่ 2 จากการเปรียบเทียบค่าในกลุ่มแรกมีค่าน้อยกว่า จะนำค่าของข้อมูลในกลุ่มแรกบันทึกในสื่อเป็นชุดของข้อมูลใหม่ที่เป็นผลลัพธ์

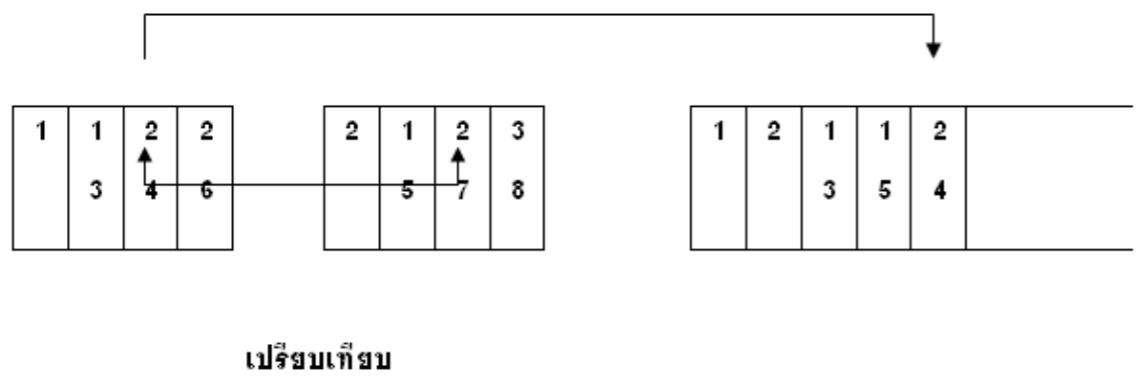


ต่อจากนั้นเลื่อนข้อมูลในกลุ่มแรกเพื่อเปรียบเทียบกับข้อมูลในกลุ่มที่ 2 ต่อไป ในกลุ่มแรกคือค่า 13 ซึ่งมีค่ามากกว่า แสดงว่าข้อมูลผลลัพธ์ตัวถัดไปคือข้อมูลในกลุ่มที่ 2 ในที่นี้คือค่า 2 จะนำค่านี้ไปบันทึกในชุดข้อมูลผลลัพธ์ดังภาพ

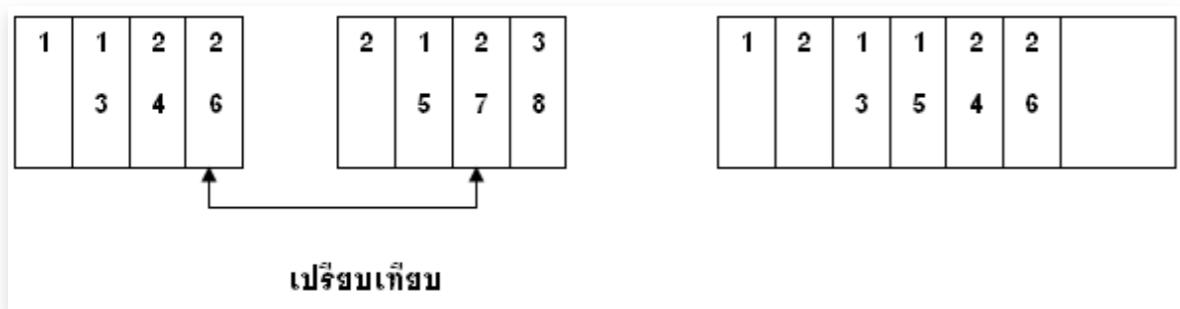


การเปรียบเทียบข้อมูลในแต่ละกลุ่มจะเลื่อนตำแหน่งของข้อมูลที่ละตำแหน่ง การเปรียบเทียบจะเปรียบเทียบเป็นคู่ๆ จนกระทั่งหมดข้อมูลกลุ่มใดกลุ่มหนึ่ง ข้อมูลที่เหลือจะบันทึกในชุดข้อมูลผลลัพธ์ ผลจากการทำงานจะทำให้ได้กลุ่มของข้อมูลที่เรียงลำดับเรียบร้อยแล้ว โดยข้อมูลทั้งหมดจะบันทึกในแฟ้มซึ่งเก็บในสื่อภายนอก

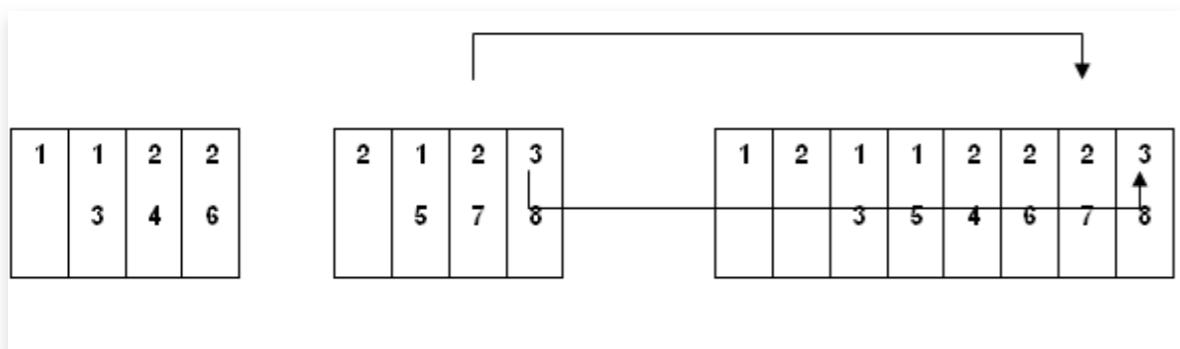




สำหรับประสิทธิภาพของการเรียงลำดับข้อมูลแบบ Merge Sort นั้น เนื่องจากว่าได้มีการแบ่งข้อมูลออกเป็นกลุ่มย่อยต่างๆ แล้วจึงค่อยทำการจัดเรียงข้อมูลกลุ่มย่อยก่อน ทำให้การเปรียบเทียบค่าลดจำนวนครั้งลงจึงทำให้ประสิทธิภาพของการจัดเรียงแบบ Merge Sort คือ $n(\log n)$



เมื่อข้อมูลกลุ่มได้กลุ่มนั่นหมายความ แต่อีกกลุ่มยังเหลืออยู่ ก็จะนำข้อมูลที่เหลือบันทึกต่อท้ายในชุดข้อมูลผลลัพธ์ดังภาพ



ตารางเปรียบเทียบประสิทธิภาพการเรียงลำดับวิธีต่างๆ

วิธีการเรียง ลำดับ	กรณีดีที่สุด	กรณีเฉลี่ย	กรณีเลวร้าย สุด
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Insertion Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Bubble Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Quick Sort	$O(n \log n)$	$O(n \log n)$	$O(n^2)$
Merge Sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$

Section 4

แบบฝึกหัด

1. ในอาร์เรย์หนึ่งประกอบด้วยสมาชิกดังนี้

3 13 7 26 44 23 98 57

อยากรับว่าหลังจากผ่านการจัดเรียงลำดับในรอบที่ 3 และ การเรียงลำดับข้อมูลในข้อมูลชุดนี้จะประกอบด้วยสมาชิกได้ บ้าง ที่เป็นไปตามอัลกอริธึม Insertion Sort

2. ในอาร์เรย์หนึ่งประกอบด้วยสมาชิกดังนี้

7 8 26 44 13 23 98 57

อยากรับว่าหลังจากผ่านการจัดเรียงลำดับในรอบที่ 4 และ การเรียงลำดับข้อมูลในข้อมูลชุดนี้จะประกอบด้วยสมาชิกได้ บ้าง ที่เป็นไปตามอัลกอริธึม Selection Sort

3. ในอาร์เรย์หนึ่งประกอบด้วยสมาชิกดังนี้

7 8 26 44 13 23 57 98

อยากรับว่าหลังจากผ่านการจัดเรียงลำดับในรอบที่ 2 และ การเรียงลำดับข้อมูลในข้อมูลชุดนี้จะประกอบด้วยสมาชิกได้ บ้าง ที่เป็นไปตามอัลกอริธึม Bubble Sort



Searching



วัตถุประสงค์

- 1.เข้าใจหลักการของ Sequential Search และ Binary Search
- 2.เปรียบเทียบความแตกต่างระหว่างการค้นหาข้อมูล ในแต่ละวิธีได้
- 3.บอกวิธีการแก้ปัญหาการชนกันของคีย์ได้

Section 1

การค้นหาข้อมูล (Searching)

เป็นกระบวนการที่สำคัญยิ่ง ในการประมวลผลข้อมูล ด้วยคอมพิวเตอร์ การค้นหาข้อมูล ในปัจจุบัน ยังคงเป็นปัญหา และยังมีการศึกษาค้นคว้า ตัวอย่างการค้นหาข้อมูล ที่พ่อจะ คุ้นเคย

ได้แก่ การค้นหาเส้นทางที่สั้นที่สุดสำหรับการเดินทาง การค้นหาทางเดินของหุ่นยนต์

การค้นหาทางเดินของหมากรุก หรือ พัลซัล (Puzzle) ต่าง ๆ การค้นหาลักษณะของหลักการกลายพันธุ์ (Genetic Algorithm)

การค้นหาความเหมือนกันของรูปภาพ หรือการรู้จำ เป็นต้น

สิ่งเหล่านี้ล้วนเป็นความท้าทาย ในการค้นหา นอกจากนี้ ในปัจจุบันยังได้นำความรู้ทางด้านสถิติและระบบผู้เชี่ยวชาญ มาช่วยในการค้นหา และการพยากรณ์เหตุการณ์ที่จะเกิดขึ้น ในอนาคต เช่น ในสาขาวิชา เดตามайнิ่ง (Data Mining)

สำหรับการค้นหาที่จะศึกษาในบทนี้ เป็นเพียงการค้นหา ข้อมูลเบื้องต้น การค้นหาเพื่อหาระเบียน หรือ เขตข้อมูล ที่มี



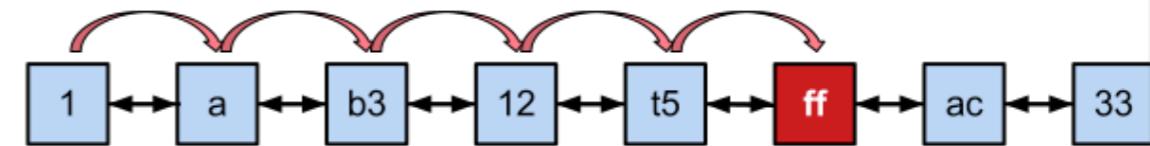
การจัดเก็บ ในแฟ้มข้อมูล หรือ ในแ阁ล์ดับ มีหรือไม่ ซึ่งในการค้นหา จะต้องมีการกำหนด เขตข้อมูล เพื่อทำหน้าที่เป็นคีย์หลัก หรือด้ชนี ในการค้นหา ซึ่งอาจเป็น รหัส ชื่อ-สกุล หรือ เขตข้อมูลอื่นๆ ที่สนใจ และสามารถระบุระเบียนข้อมูลได้ การค้นหาจะหาได้รวดเร็ว หรือมีประสิทธิภาพเพียงใด ขึ้นอยู่กับ หลักการ การค้นหาเป็นสำคัญ สำหรับขั้นตอนวิธีการค้นหา ข้อมูลเบื้องต้น ที่จะกล่าวถึงนี้ คิดว่าจะเป็นประโยชน์ในการใช้งาน และเป็นพื้นฐานในการเรียนรู้ต่อไป

อัลกอริทึมเพื่อการค้นหาขั้นพื้นฐาน ประกอบด้วย 3 วิธี

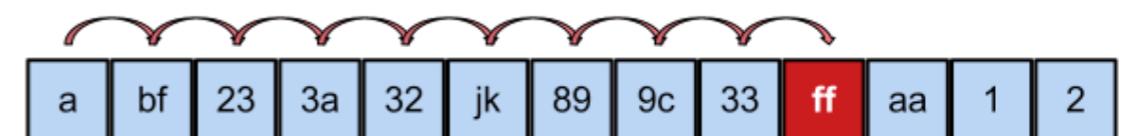
1. การค้นหาแบบลำดับ (Sequential Search)
2. การค้นหาแบบไบนารีทรี (Binary Search)
3. การค้นหาข้อมูลแบบแฮชชิ่ง (Hashing Search)

Search

1. Linked Lists search for "ff"



2. Arrays sequential search



!Ineffective operation

ภาพแสดงการค้นหาข้อมูลด้วยวิธีการแบบลำดับ

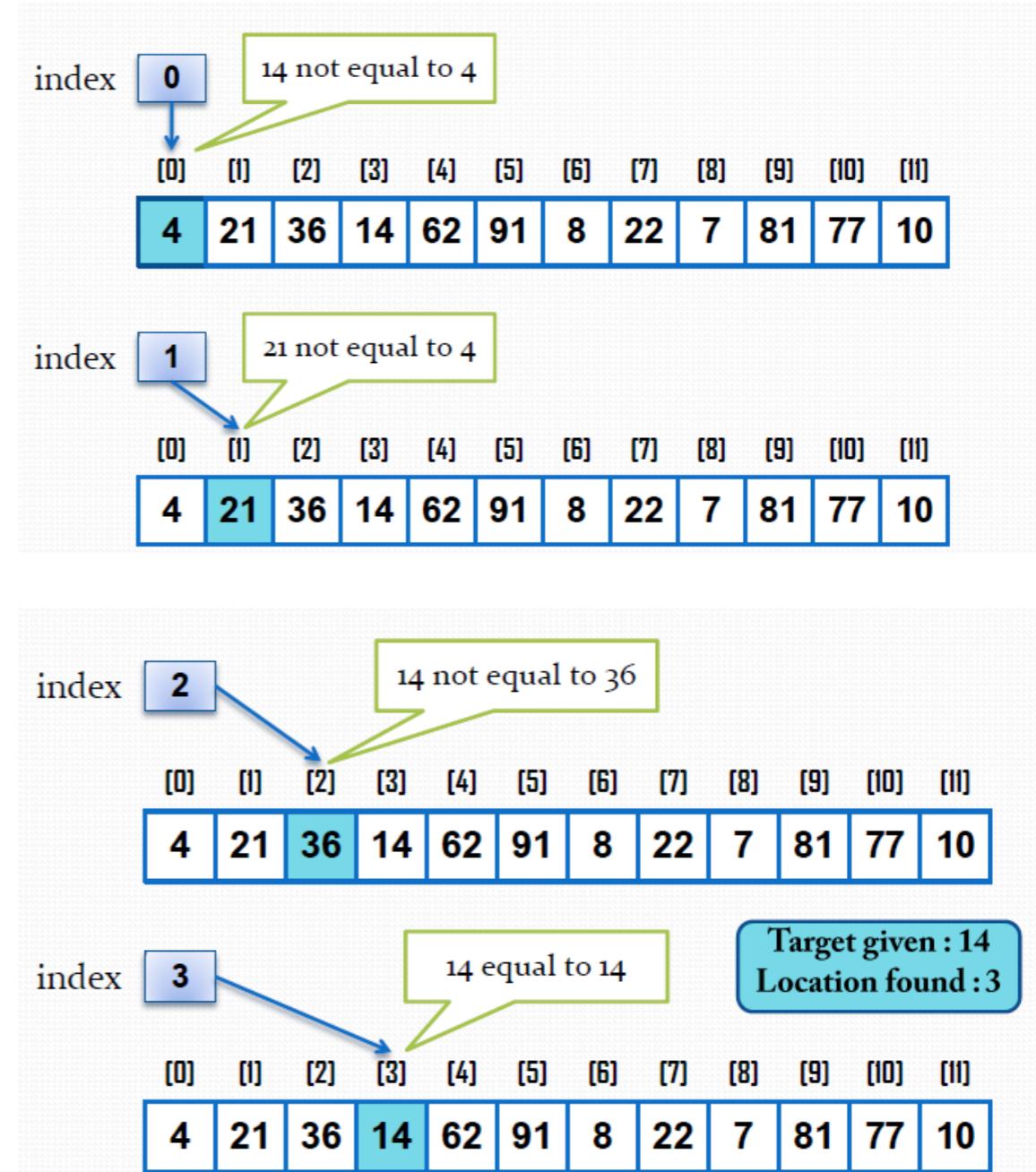
Section 2

การค้นหาข้อมูลแบบลำดับ (Sequential Search)

การค้นหาข้อมูลแบบลำดับ หรือเรียกว่าการค้นหาข้อมูลแบบเชิงเส้น (Linear Search)

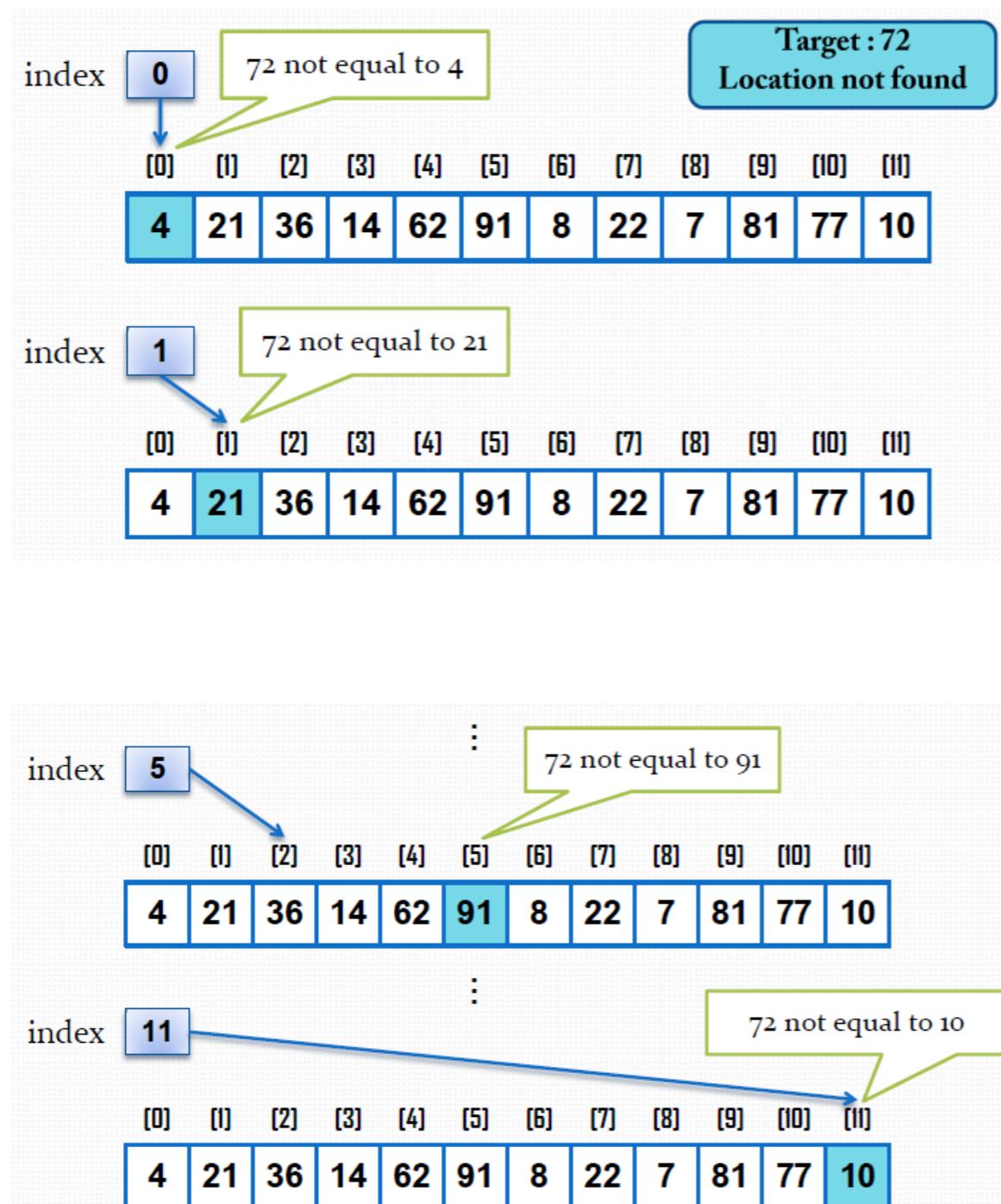
- ❖ นำไปใช้งานบนลิสต์ที่ไม่ได้มีการเรียงลำดับข้อมูล
- ❖ **มีหลักการในการค้นหา** คือ จะนำค่าที่ต้องการค้นหา (Target) ไปเปรียบเทียบกับข้อมูลภายในลิสต์ที่ลงทะเบียนไว้พบ ก็จะเปรียบเทียบกับข้อมูลตัวถัดไปเรื่อยๆ จนพบข้อมูลที่ต้องการหรืออาจเปรียบเทียบจนถึงข้อมูลตัวสุดท้ายแล้วไม่พบ

ตัวอย่าง การค้นหาแบบเรียงลำดับบนลิสต์ที่ไม่ได้เรียงลำดับข้อมูลในกรณี ค้นหาพบ (Target = 14)



ภาพแสดงการค้นหาข้อมูลบนลิสต์ที่ไม่ได้เรียงลำดับและค้นหาพบ

ตัวอย่าง การค้นหาแบบเรียงลำดับบนลิสต์ที่ไม่ได้เรียงลำดับข้อมูลในกรณีค้นหาไม่พบ (Target = 72)



ภาพการค้นหาข้อมูลบนลิสต์ที่ไม่ได้เรียงลำดับ และไม่พบข้อมูล

การค้นหาแบบเรียงลำดับจะหมายความว่า การค้นหาค่าในชุดข้อมูลที่ไม่ได้เรียง

แต่ถ้าข้อมูลเรียบร้อยแล้ว จะมีข้อเสีย คือ เมื่อค้น Target ในชุดข้อมูลจนถึงตัวที่มีค่ามากกว่า Target แล้ว การค้นหายังไม่ยุติการค้นยังคงวนรอบเพื่อเปรียบเทียบข้อมูลจนถึงตัวสุดท้ายในชุดข้อมูล ทำให้เสียเวลา

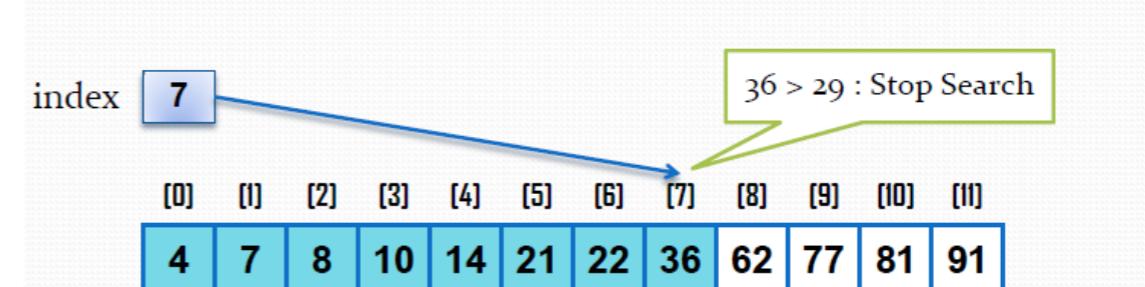
การปรับปรุงประสิทธิภาพการค้นหาแบบลำดับ

การค้นหาข้อมูลแบบลำดับด้วยเทคนิคการเรียงลำดับข้อมูล เรียงลำดับข้อมูลจากน้อยไปมาก

การค้นหาข้อมูลแบบลำดับด้วยเทคนิค Self Reordering

ปรับปรุงขั้นตอนวิธีการค้นหา โดยการค้นหาข้อมูลจะหยุดทำการค้นหาเมื่อพบว่า Target ที่ใช้ค้นมีค่าน้อยกว่าข้อมูลในชุดข้อมูล

Target = 29



ภาพแสดงการค้นหา จะหยุดการค้นหาทันทีที่ค่าที่ค้นหาน้อยกว่าชุดข้อมูล

Section 3

การค้นหาข้อมูลแบบไบナรี (Binary Search)

เป็นการค้นหากับข้อมูลที่ถูกเรียงลำดับแล้ว เหมาะกับ การค้นหาข้อมูลที่มีปริมาณข้อมูลจำนวนมาก

ขั้นตอนการค้นหา

- ❖ แบ่งลิสต์ออกเป็น 2 ส่วน
- ❖ กำหนดค่า Target หรือค่าที่ต้องการค้นหา
- ❖ ทำการเปรียบเทียบข้อมูลในลิสต์โดยทำการแบ่งครึ่ง ลงไปเรื่อยๆ จนกว่าจะพบค่า Target หรือ ไม่พบค่า Target ในลิสต์

การค้นหาตำแหน่งกึ่งกลางของลิสต์ จะเป็นต้องใช้ ตัวแปร 3 ตัวด้วยกันคือ

1. ตัวแปร begin

2. ตัวแปร mid

3. ตัวแปร end

การคำนวณตำแหน่งกึ่งกลางของลิสต์จะเป็นไปตามสูตร ดังนี้

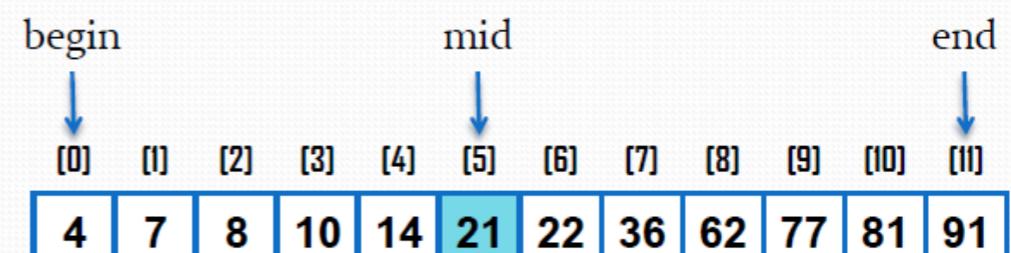
$$mid = [(begin + end) / 2]$$

ตัวอย่าง มีชุดตัวเลขที่เรียงลำดับแล้วภายในลิสต์ คือ {4, 7, 8, 10, 14, 21, 22, 36, 62, 77, 81, 91} และค่า Target ที่ต้องการค้นหาคือค่า 22

รอบที่ 1

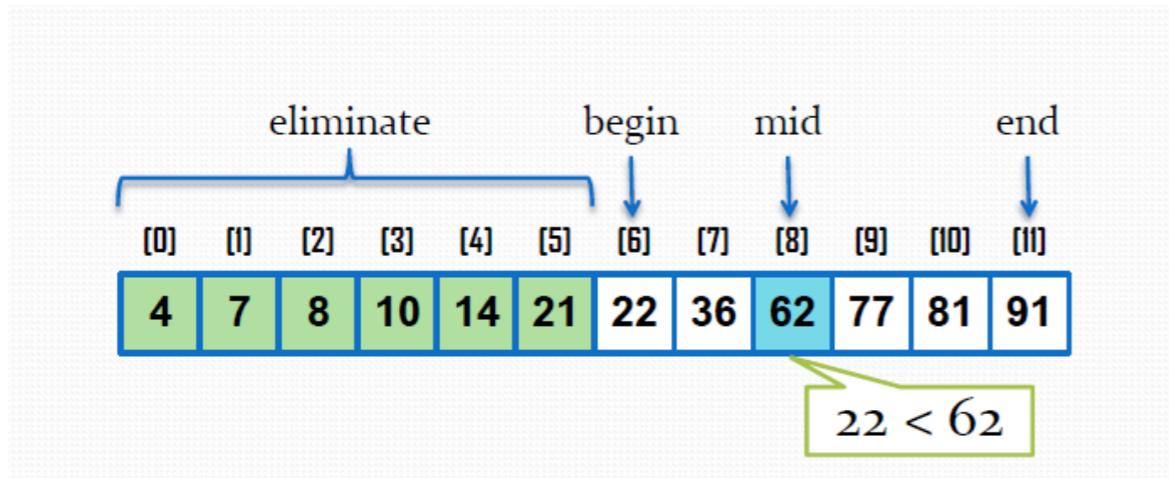
การคำนวณหาตำแหน่งกึ่งกลาง $= (0+11) / 2 = 5$ (ให้ปัดเศษทิ้ง)

การคำนวณหาตำแหน่งกึ่งกลาง $= (0+11) / 2 = 5$ (ให้ปัดเศษทิ้ง)



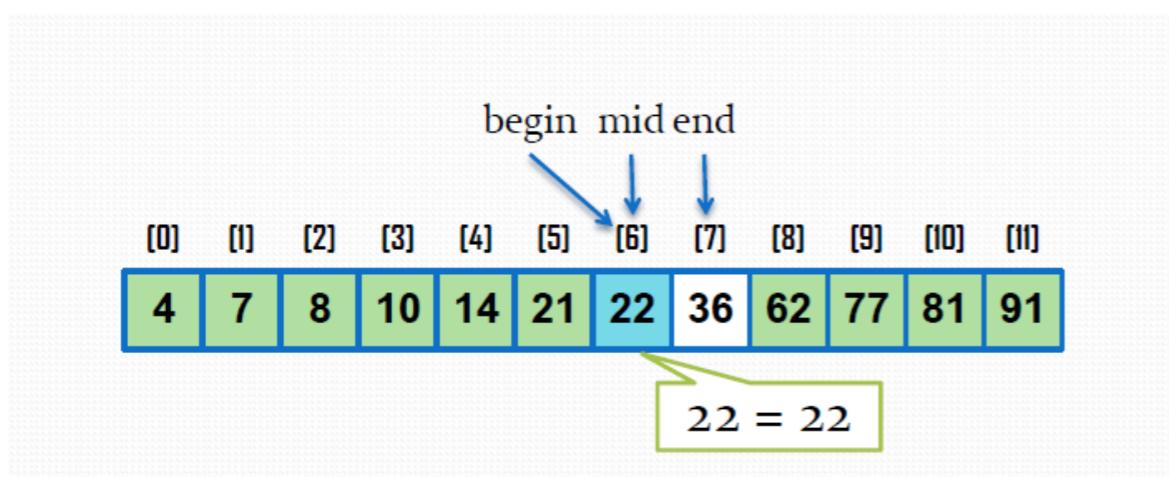
รอบที่ 2

$$\text{การคำนวณหาตำแหน่งกึ่งกลาง} = (6+11) / 2 = 8$$



รอบที่ 3

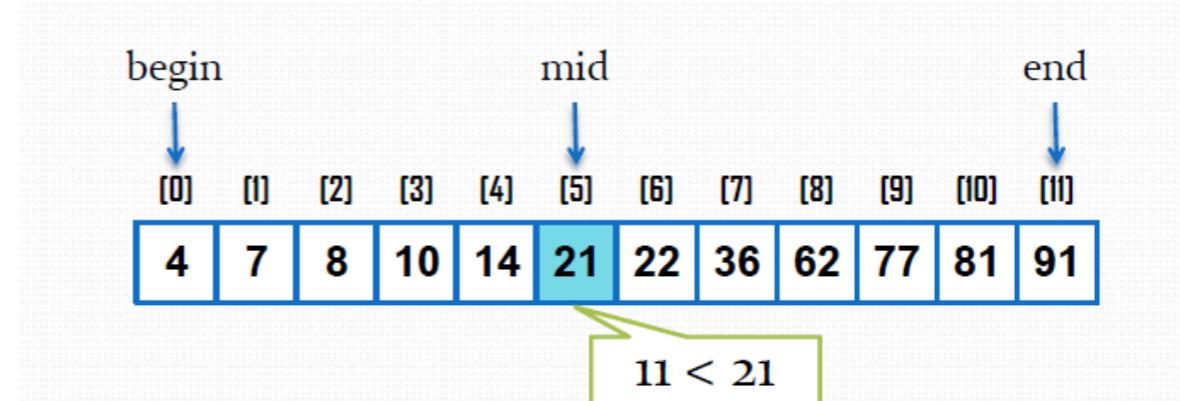
$$\text{การคำนวณหาตำแหน่งกึ่งกลาง} = (6+7) / 2 = 6$$



ตัวอย่าง มีชุดตัวเลขที่เรียงลำดับแล้วภายในลิสต์ คือ {4, 7, 8, 10, 14, 21, 22, 36, 62, 77, 81, 91} และค่า Target ที่ต้องการค้นหาคือค่า 11

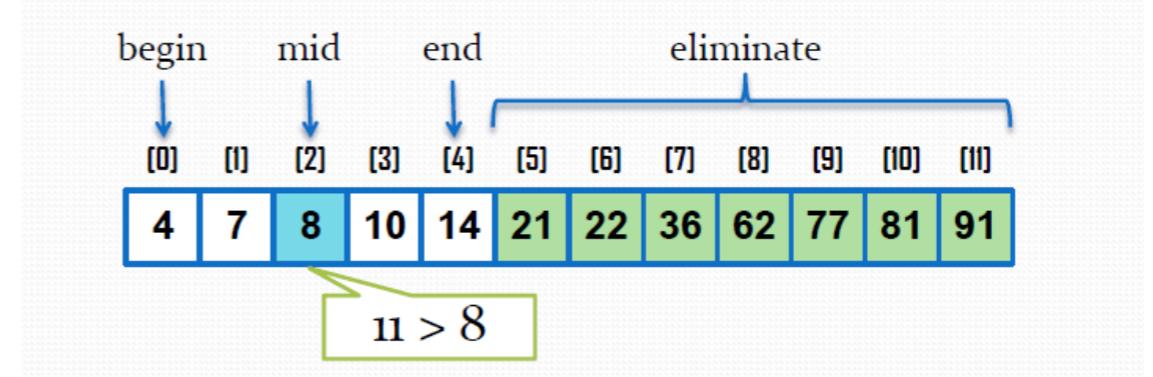
รอบที่ 1

$$\text{การคำนวณหาตำแหน่งกึ่งกลาง} = (0+11) / 2 = 5 \text{ (ให้ปัดเศษทิ้ง)}$$



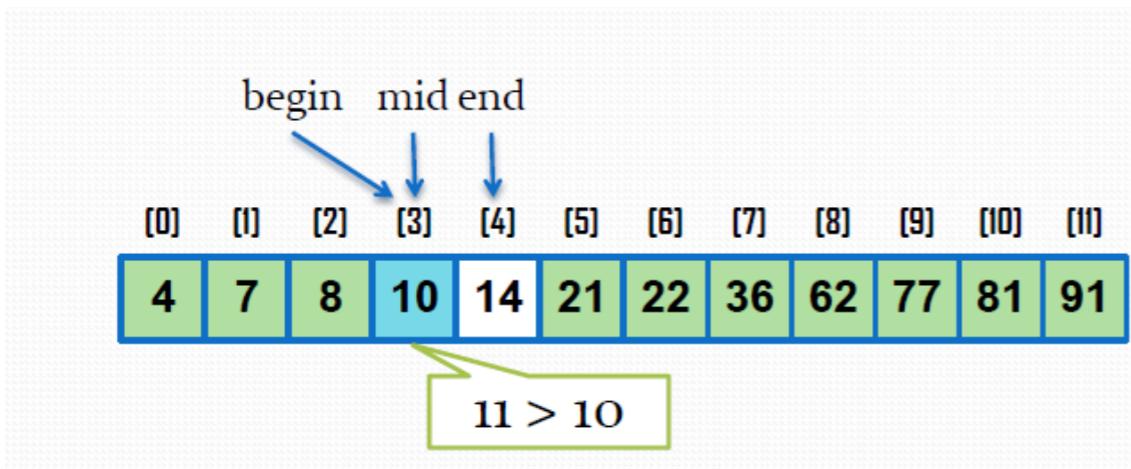
รอบที่ 2

$$\text{การคำนวณหาตำแหน่งกึ่งกลาง} = (0+4) / 2 = 2$$



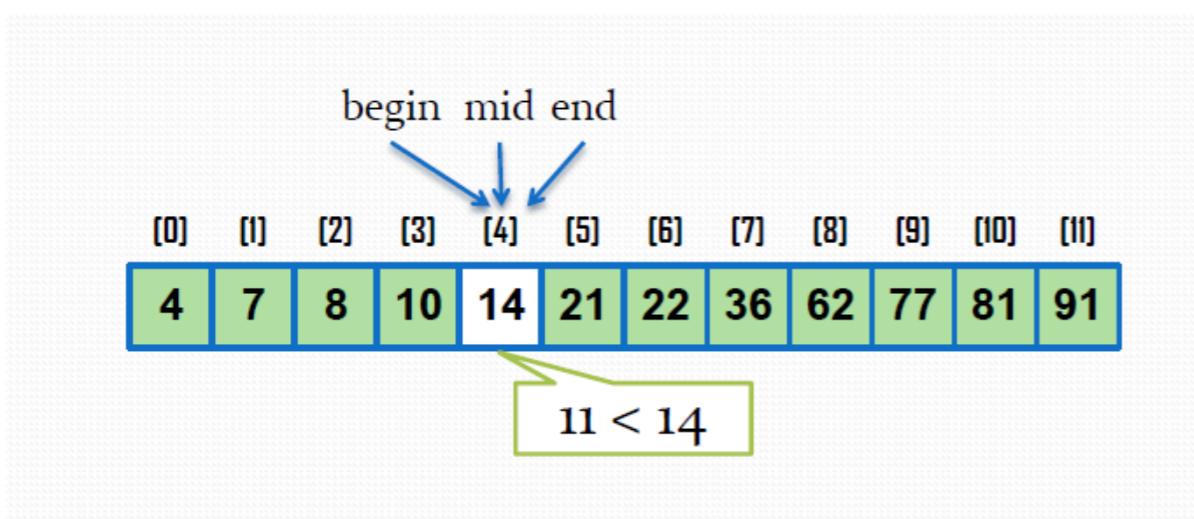
รอบที่ 3

$$\text{การคำนวณหาตำแหน่งกึ่งกลาง} = (3+4) / 2 = 3$$



รอบที่ 4

$$\text{การคำนวณหาตำแหน่งกึ่งกลาง} = (4+4) / 2 = 4$$

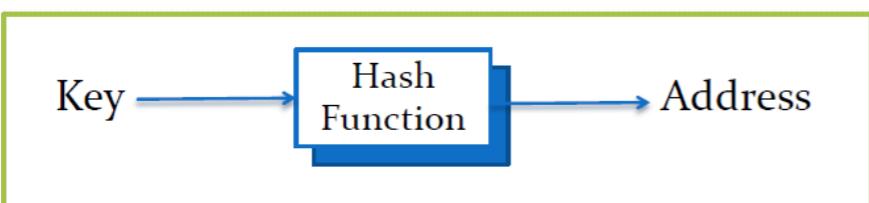


Section 4

การค้นหาข้อมูลแบบแฮชชิ่ง (Hashing Search)

- ❖ จะมีคีย์ (Key) เพื่อใช้เป็นตัวค้นหา
- ❖ ซึ่งจะนำคีย์ไปผ่านฟังก์ชันแฮช (Hash Function) เพื่อได้มาซึ่งตำแหน่งข้อมูล (Address)
- ❖ ตำแหน่งที่ได้เรียกว่า Home Address

การแปลงคีย์ ให้เป็น Address คือ การแปลงข้อมูลให้ไปอยู่ในตาราง Address ที่เตรียมไว้ ตารางนี้เรียกว่า ตารางแฮช (Hash Table)



ภาพแสดงวิธีการแปลงคีย์ให้เป็น Address

คีย์ (Key)

คือ ข้อมูลที่ต้องการนำไปค้นหา ซึ่งค่าของคีย์จะไม่มีการซ้ำกัน

ฟังก์ชันแฮช (Hash Function)

คือ สูตรหรือฟังก์ชันที่ใช้ส่าหรับแปลงคีย์ให้เป็นตำแหน่งแอดเดรส เมื่อได้แอดเดรสแล้ว ก็จะสามารถหาแอดเดรสนี้เข้าถึงตำแหน่งข้อมูลที่ต้องการในตารางแฮชได้

ตารางแฮช (Hash Table)

คือ ตารางที่ใช้ส่าหรับเก็บข้อมูล

วิธีการหาร (Modulo-Division Method)

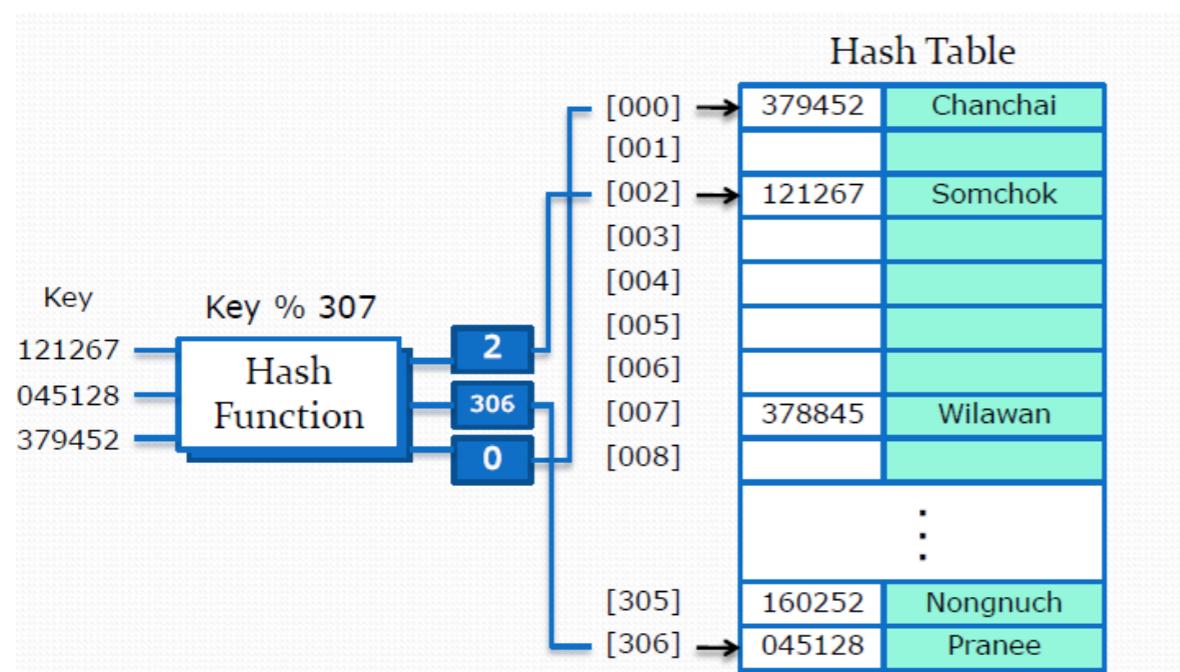
หารเพื่อนำเศษที่ได้จากการหารมาเป็นตำแหน่งแอดเดรสในตารางแฮช มีสูตรการคำนวณดังนี้

$$\text{Address} = \text{key MODULO listSize}$$

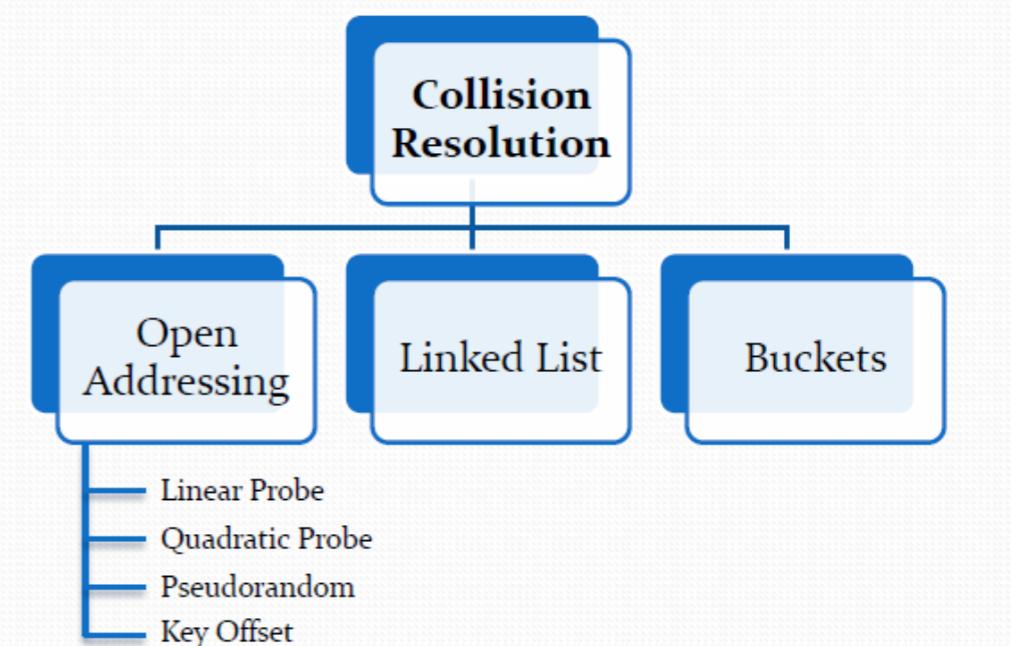
ขนาดของตารางแฮชจะขึ้นอยู่กับความต้องการของผู้ใช้งาน ทั่วไปมักมีการกำหนดตารางแฮชให้มีขนาดใหญ่กว่าจำนวนคีย์ที่มีอยู่ เพื่อป้องกันการชนกันของคีย์น้อยที่สุด

ตัวอย่าง บริษัทแห่งหนึ่งมีจำนวนพนักงานมากกว่า 100 คน และเป็นบริษัทที่กำลังเติบโต จึงมีแผนการว่าจะรับพนักงานเพิ่มขึ้น จึงมีการจัดเตรียมพื้นที่รองรับจำนวนพนักงานเป็น 300 คน ในที่นี้ได้ทำการคัดเลือกตัวเลขที่มากกว่า 300 คือ 307 มาเป็นตัวหาร จึงได้อาร์เรย์ขนาด 307 ช่อง เพื่อใช้สำหรับรองรับและเดรสตั้งแต่ 0 ถึง 306

การแฮชซิงด้วยวิธีการหาร (Modulo-Division Hashing)



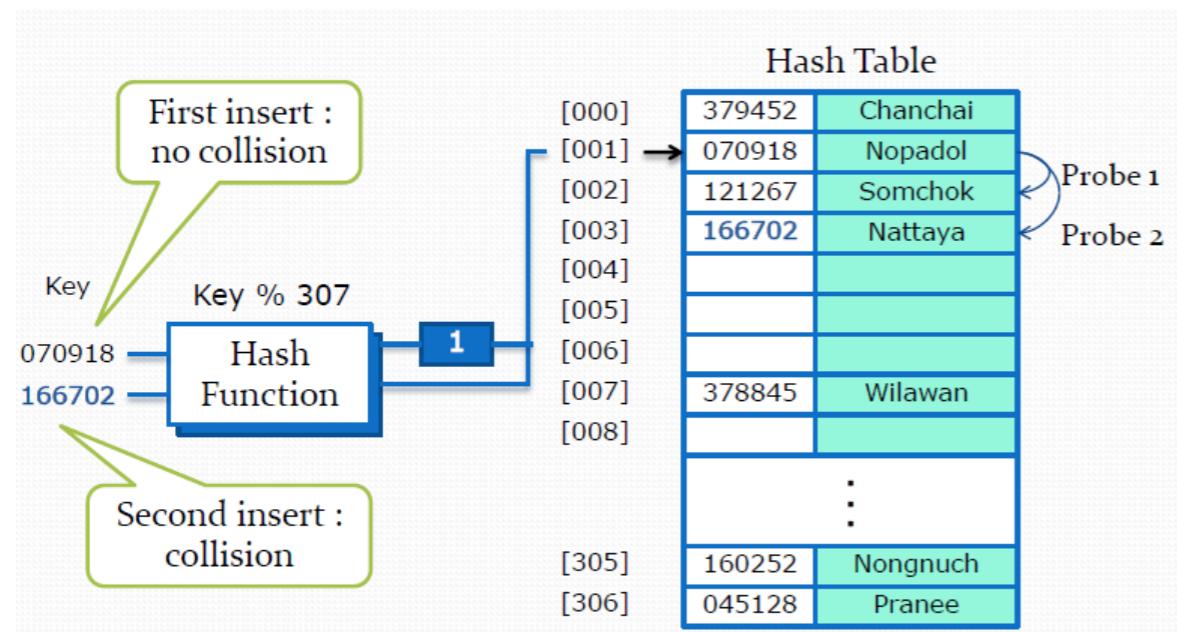
แนวทางแก้ไขเมื่อมีการชนกันของคีย์ (Collision Resolution)



การแก้ปัญหาการชนกันของคีย์แบบ Open Addressing ด้วยวิธีลิнейร์เพอบ (Linear Probe)

- ❖ เมื่อข้อมูลไม่สามารถจัดเก็บในตำแหน่งเดรส์เดิม ข้อมูลจัดเก็บอยู่ก่อน)
- ❖ แก้ไขการชนกันด้วยการนาแอดเดรสปั๊จจุบันมากกว่าเพิ่มอีกหนึ่งตำแหน่ง

- ❖ ก็จะเป็นตำแหน่งถัดไป แต่ถ้าไม่ว่างอีกต้องทำการ Probe ในรอบที่สอง ด้วยการบวกเพิ่มอีกหนึ่งตำแหน่ง เพื่อหาแอดเดรสที่ว่าง



ภาพแสดงวิธีการแก้ด้วยวิธี Open Addressing

ข้อดี

- ❖ เป็นวิธีที่ง่าย
- ❖ ข้อมูลที่แทรกเข้าไปใหม่ (กรณีคีย์ชนกัน) จะอยู่ใกล้ตำแหน่งแอดเดรสจริงของตัวมันเอง

ข้อเสีย

- ❖ พังก์ชันแฮชที่ดี จะต้องออกแบบให้คีย์มีการกระจายสม่ำเสมอ เพื่อลดการชนกันของคีย์ให้มากที่สุด

- ❖ เมื่อเกิดการชนกันของคีย์ ก็จะมีความพยายามแทรกข้อมูลในตำแหน่งว่างถัดไปจากแอดเดรสจริง ส่งผลให้เกิดการรวมกลุ่มของข้อมูลมากขึ้น

- ❖ ส่งผลต่อการกระจายคีย์ในตารางแฮชสະດุดลง (คีย์ชนกันมากขึ้น)

การแก้ปัญหาการชนกันของคีย์แบบ Linked List หรือ Chaining

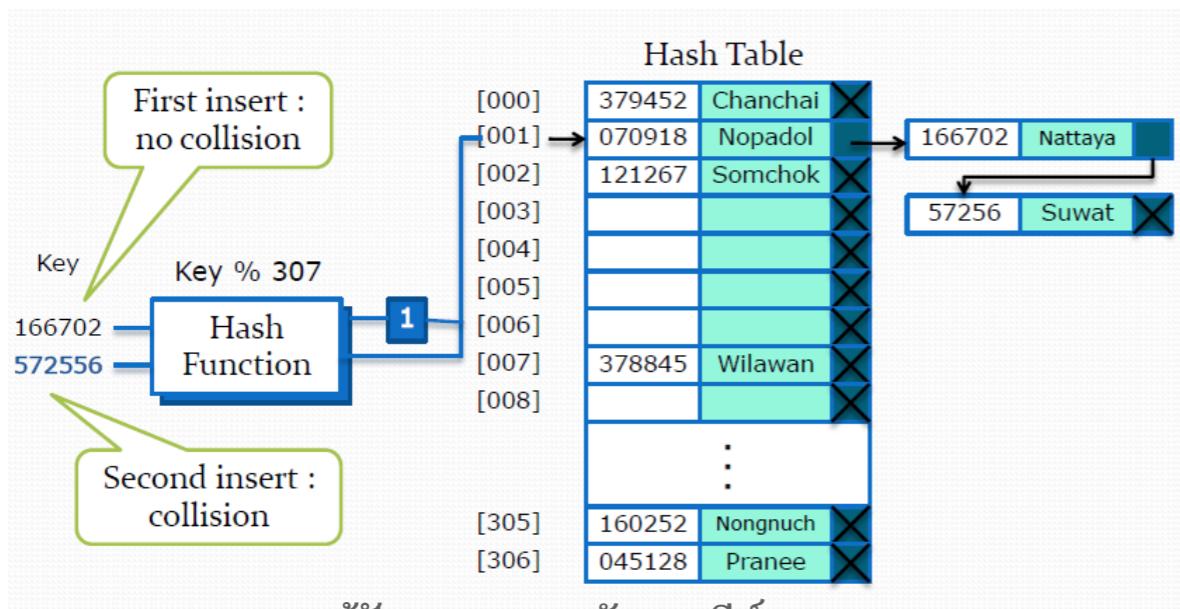
- ❖ เมื่อคีย์ได้ผ่านฟังก์ชันแฮชแล้วเกิดการชนกันของตำแหน่งแอดเดรสในตารางแฮช

- ❖ จะมีการใช้ลิงก์ลิสต์เป็นตัวเชื่อมโยงถัดไปเป็นลูกโซ่

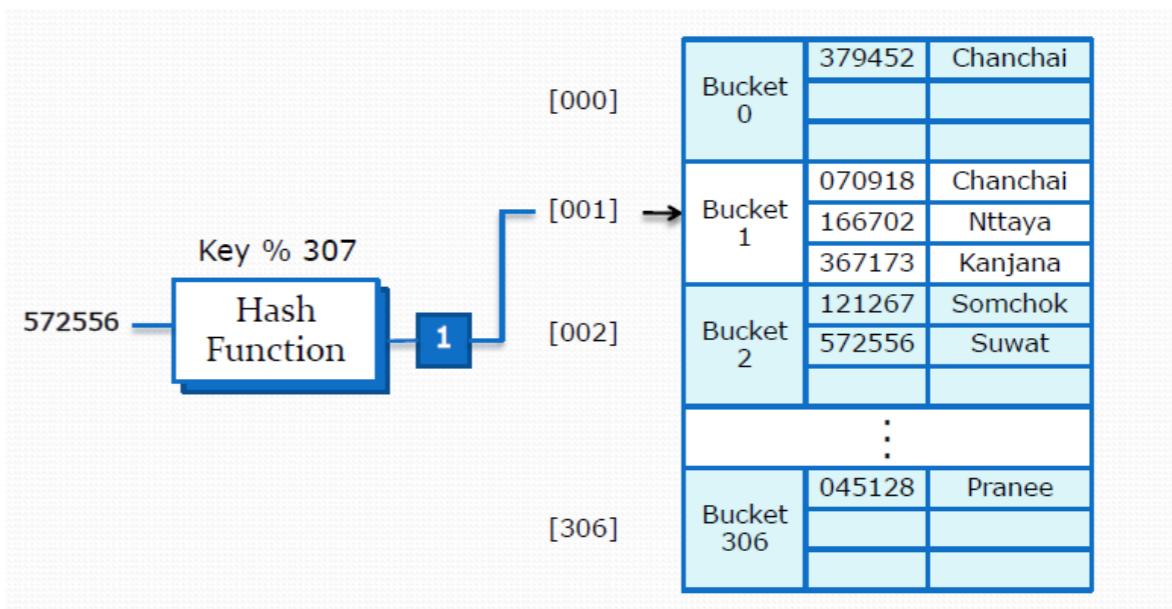
- ❖ จะมีการใช้พื้นที่เพื่อจัดเก็บข้อมูลที่ชนกันแยกออกจากต่างหากด้วยการใช้โครงสร้างข้อมูลแบบลิงก์ลิสต์มีพื้นที่สองส่วน คือ

- ❖ พื้นที่หลัก (prime Area) ใช้บรรจุ Home Address ทั้งหมด

- ❖ พื้นที่ส่วนโอเวอร์โฟล์ (Overflow) ใช้เก็บข้อมูลที่มีแอดเดรสเดียวกัน



ภาพแสดงการแก้ปัญหาการชนกันของคีย์แบบ Linked list



ภาพแสดงการแก้ปัญหาการชนกันของคีย์แบบ Bucket

การแก้ปัญหาการชนกันของคีย์แบบ Bucket Hashing

- ❖ เมื่อคีย์ได้ลูกแผลงใน Bucket ที่เสมอตະกร้า
- ❖ คีย์ที่ชนกันยังสามารถบรรจุลงในตารางແ驿ชร่วมกันในตະกร้า
- ❖ มีการจัดสรรข้อมูลตามแน่งที่จัดเก็บในรูปแบบของตารางหลายช่อง
- ❖ หากมีการชนกันของคีย์อีก ก็จะจัดเก็บลงในตารางตามแน่งถัดไปจนกระทั่งเต็ม

จากการศึกษาเกี่ยวกับการค้นหาข้อมูลด้วยวิธีแฮชชิง จะเห็นถึงปัญหาที่เกิดขึ้นได้จากวิธีนี้คือ การกันของคีย์ แต่ก็มีแนวทางในการแก้ไขปัญหาที่หลากหลาย ดังนั้นในการทำงานจริง เราสามารถนำหลาย ๆ วิธีข้างต้นมาประยุกต์ใช้งานร่วมกันได้ ตัวอย่างเช่น ฐานข้อมูลที่สร้างขึ้นด้วยวิธี Bucket ถ้า Bucket เต็มก็สามารถหาวิธี Linear Probe เข้ามาแก้ไข หรืออาจจะใช้วิธีลิงค์ลิสต์ก็ได้

แบบฝึกหัด

1. จงอธิบายหลักการค้นหาข้อมูลแบบลำดับมาให้พอเข้าใจ
2. จงสรุปหลักการค้นหาข้อมูลแบบใบเสร็จมาพอเข้าใจ
3. สมมติว่ามีลิสต์หนึ่งประกอบด้วยสมาชิก 100 101 154 160 188 205 599 1020 2002 และหากค่า Target ที่ต้องการคือ 154 ให้แสดงขั้นตอนการหาแบบใบเสร็จ
4. จากข้อที่ 3 หากค่า Target ที่ต้องการคือ 98 ให้แสดงขั้นตอนการหาแบบใบเสร็จ