- 418531 Data Structures & Algorithm Analysis

⇒ วิชานี้เกี่ยวกับ "อัลกอริทึม"

- อัลกอริทึมคืออะไร?

⇒ คำตอบของ "ปัญหาทางการคำนวณ" (computational problem)

- ปัญหาทางการคำนวณคืออะไร?

⇒ คำถามว่าเราจะคำนวณผลลัพธ์ "อย่างไร"?

⇒ ไม่เน้นคำตอบ

⇒ เน้นวิธีการ

⇒ วิธีการต้องชัดเจน สามารถให้คอมพิวเตอร์นำไปปฏิบัติได้!

- ส่วนประกอบของปัญหาการคำนวณ

⇒ ข้อมูลเข้า (input)

⇒ ข้อมูลออก (output)

⇒ เงื่อนไข = ความสัมพันธ์ระหว่างข้อมูลเข้า กับข้อมูลออก

- ตัวอย่าง: การหาค่ามากสุด

⇒ ข้อมูลเข้า: จำนวน $a_1, a_2, \ldots, a_n$

⇒ ข้อมูลออก: เลข m

⇒ เงื่อนไข: $m = a_k$ บางตัว และ

$m \geq a_i$ สำหรับ i ทุกตัว

- ตัวอย่าง: การใช้งานจำนวนเฉพาะ:

⇒ ข้อมูลเข้า: จำนวนเต็มบวก N

⇒ ข้อมูลออก: YES/NO

⇒ เงื่อนไข: ตอบ YES ถ้า N เป็นจำนวนเฉพาะ มิเช่นนั้นตอบ NO

- ตัวอย่าง: การเรียงลำดับข้อมูล

⇒ ข้อมูลเข้า: จำนวน $a_1, a_2, \ldots, a_n$

⇒ ข้อมูลออก: ลำดับ $b_1, b_2, \ldots, b_n$

⇒ เงื่อนไข: ① $b_1 \leq b_2 \leq b_3 \leq \ldots \leq b_n$

② $b_1, b_2, \ldots, b_n$ เป็น permutation ของ $\{a_1, a_2, \ldots, a_n\}$

- โปรแกรม vs อัลกอริทึม

=> โปรแกรม = ตอบแก้ปัญหาทางการคำนวณ

=> อัลกอริทึม = วิธีการแก้ปัญหาทางการคำนวณ

=> ทำไมไม่ศึกษาโปรแกรมกันเลย?

ตอบ รายละเอียดลึกยิ่งยาก

↳ เขียนด้วยภาษาอะไร?

↳ รันบนเครื่อง คอมพิวเตอร์อะไร?

=> อัลกอริทึม — ไม่ขึ้นกับภาษา

ไม่ขึ้นกับเครื่อง

↳ แต่ต้องมีความชัดเจน พอที่จะเอา

ไปแปลงเป็นโปรแกรมได้

- ลักษณะของอัลกอริทึมที่ดี

⇒ ถูกต้อง ⟶ ต้องพิสูจน์ได้

⇒ ประหยัด

- เร็ว = ประหยัดเวลา

- ใช้หน่วยความจำน้อย.

- การวิเคราะห์อัลกอริทึม (Algorithm Analysis)

⇒ เป็นการตรวจสอบดูว่าอัลกอริทึม "ประหยัด" แค่ไหน

↳ อัลกอริทึมใช้เวลาทำงานนานแค่ไหน

↳ อัลกอริทึมใช้หน่วยความจำมากแค่ไหน

- พิจารณาปัญหาการหาค่ามากที่สุดของลำดับ

Input: อาเรย์ a ขนาด n ช่อง

Output: เลข k โดยที่ $0 \leq k < n$

เงื่อนไข: a[k] มีค่าไม่น้อยกว่าสมาชิกตัวอื่น ๆ ในอาเรย์

- เวลาบรรยายอัลกอริทึมเราทำได้หลายแบบ

1. pseudocode
2. ภาษาไทย

- อัลกอริทึมที่แก้ปัญหาการหาค่ามากที่สุด เขียนเป็น pseudocode:

```
MAX (a, n)
{
    k ← 0
    for i ← 1 to n-1 do
        if a[i] > a[k] then
            k = i
    return k
}
```

- อัลกอริทึมแก้ปัญหาการหาค่ามากที่สุดเป็นภาษาไทย

ให้ k เป็นตัวแปรที่เก็บตำแหน่งของตัวเลขที่มีค่ามากที่สุดไว้

ตอนแรกให้ k = 0 ก่อน

(เป็นการสมมติว่าเรยชองแรกใหญ่สุด)

หลังจากนั้นจึงเปรียบเทียบค่า a[k]

กับ a[1], a[2], ..., a[n-1]

ทีละตัว ถ้า a[i] > a[k]

เราก็จะเปลี่ยนให้ k = i

- เกณฑ์การออกแบบอัลกอริทึม
  1. เขียนอธิบายเป็นภาษามนุษย์ก่อน
  2. แล้วจึงเขียน pseudocode
  ถ้า 1 ไม่ชัดเจนพอ.
- อย่าเขียน pseudocode เลย
  → อ่านไม่รู้เรื่อง
- เขียนบรรยายหรืออธิบายแนวความคิดของแต่ละขั้นตอนด้วย
- การแยกอัลกอริทึมเป็นขั้นๆ ด้วย
  จะช่วยให้เข้าใจง่ายขึ้นมาก

- เราจะวิเคราะห์เวลาการทำงานของอัลกอริทึม หาค่ามากที่สุดกัน

⇒ เวลาการทำงาน = จำนวนคำสั่งที่ปฏิบัติ

⇒ เราต้องนับจำนวนคำสั่ง

```
MAX (a, n)
{
    k ← 0                          1 ครั้ง
    for i ← 1 to n-1 do            n-1
        if a[i] > a[k] then        n-1
            k = i                  ≤ n-1
    return k                       1
}
```

- ฉะนั้นโปรแกรมนี้ทำงานไม่เกิน

$$1 + (n-1) + (n-1) + (n-2) + 1$$

$$= 3n - 1 \quad \text{คำสั่ง}$$

- ข้อสังเกต:

⇒ เวลาการทำงานขึ้นอยู่กับขนาดของข้อมูลเข้า n

- เวลาการทำงานเป็นฟังก์ชัน $T(n)$
- เราได้ว่า $2n \leq T(n) \leq 3n-1$

⇒ ในการทำงานขึ้นอยู่กับข้อมูลด้วย

- ถ้า $a[0]$ ใหญ่สุด → $2n$
- ถ้าข้อมูลเรียงจากน้อยไปมาก → $3n-1$

- เวลาการทำงานแบ่งออกเป็น

⇒ best case: กรณีที่ข้อมูลทำให้อัลกอริทึมทำงานได้เร็วที่สุด

base case ของอัลกอริทึมที่แล้ว = 2n

⇒ worst case: กรณีที่ข้อมูลทำให้อัลกอริทึมทำงานได้ช้าที่สุด

worst case ของอัลกอริทึมที่แล้ว = 3n - 1

⇒ ปกติเราจะให้ T(n) แทน

worst case running time.

เนื่องจากมันรับประกันว่าจะไม่ช้ากว่านี้แล้ว

- Insertion Sort

⇒ อัลกอริทึม สำหรับการเรียงข้อมูลแบบหนึ่ง

⇒ input: อะเรย์ a ขนาด n ช่อง

⇒ ต้องการเรียงข้อมูลใน a โดยสลับที่สมาชิกต่าง ๆ

- output กับ input อยู่ในที่เดียวกัน

⇒ แนวคิด

- แบ่งอะเรย์เป็น 2 ส่วน
  ⇒ เรียงแล้ว
  ⇒ ยังไม่ได้เรียง

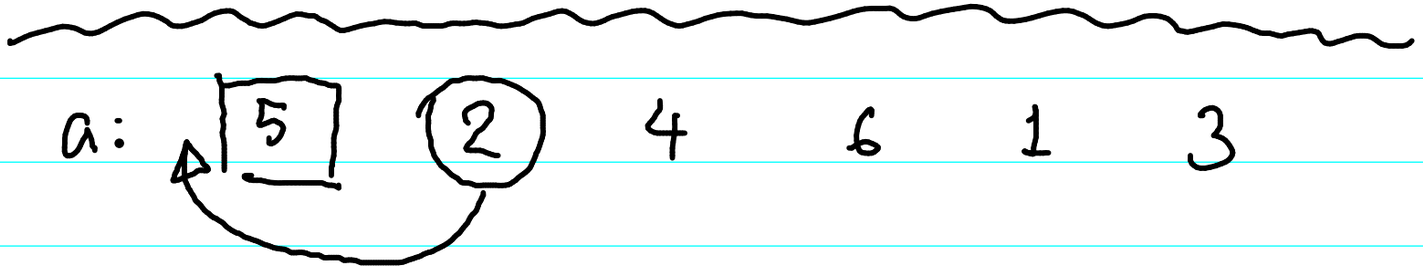
- หยิบสมาชิกส่วนยังไม่ได้เรียงมาแทรกในส่วนที่เรียงแล้ว

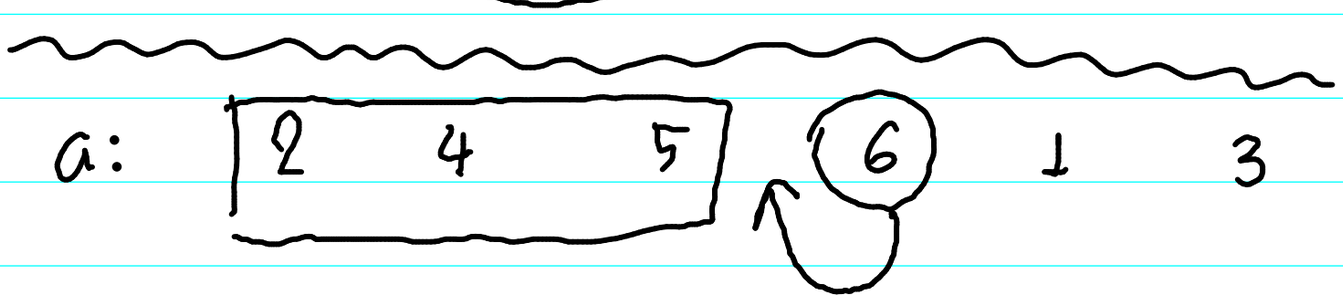
ข้อมูลเข้า

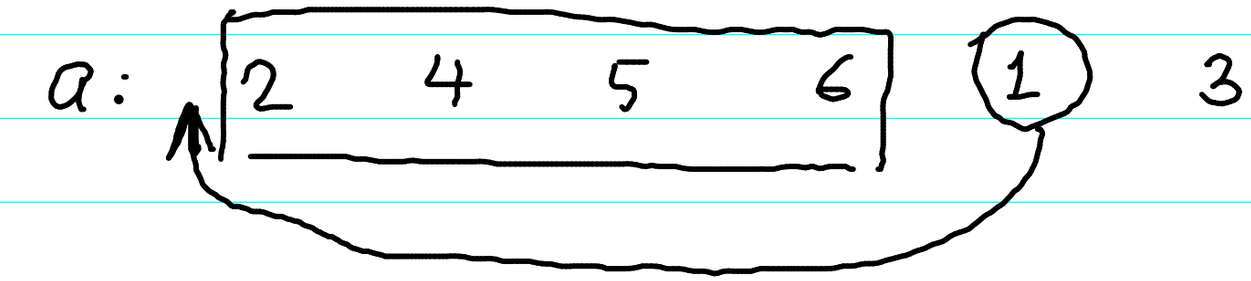
a: 5 2 4 6 1 3

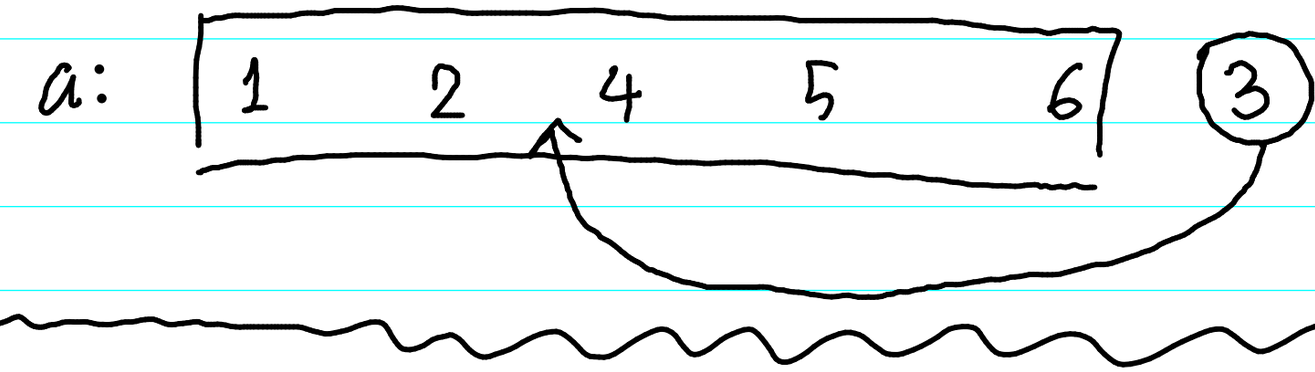
เราทำ insertion sort

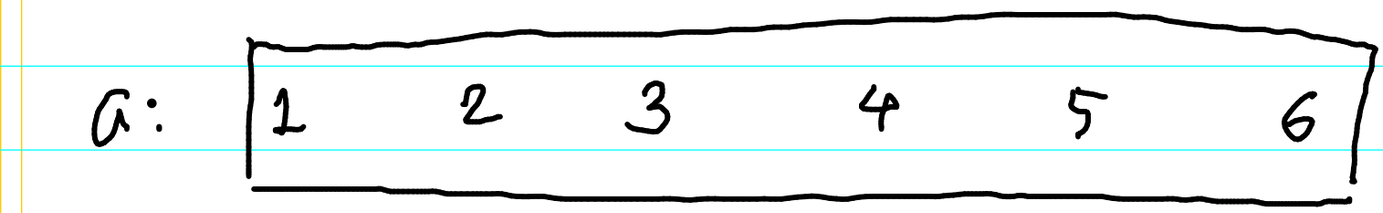
a: [5] 2 4 6 1 3






a:
2 4 5 6
1
3


a:
1 2 4 5 6
3


a:
1 2 3 4 5 6

- ลองแทรกทำอย่างไร?

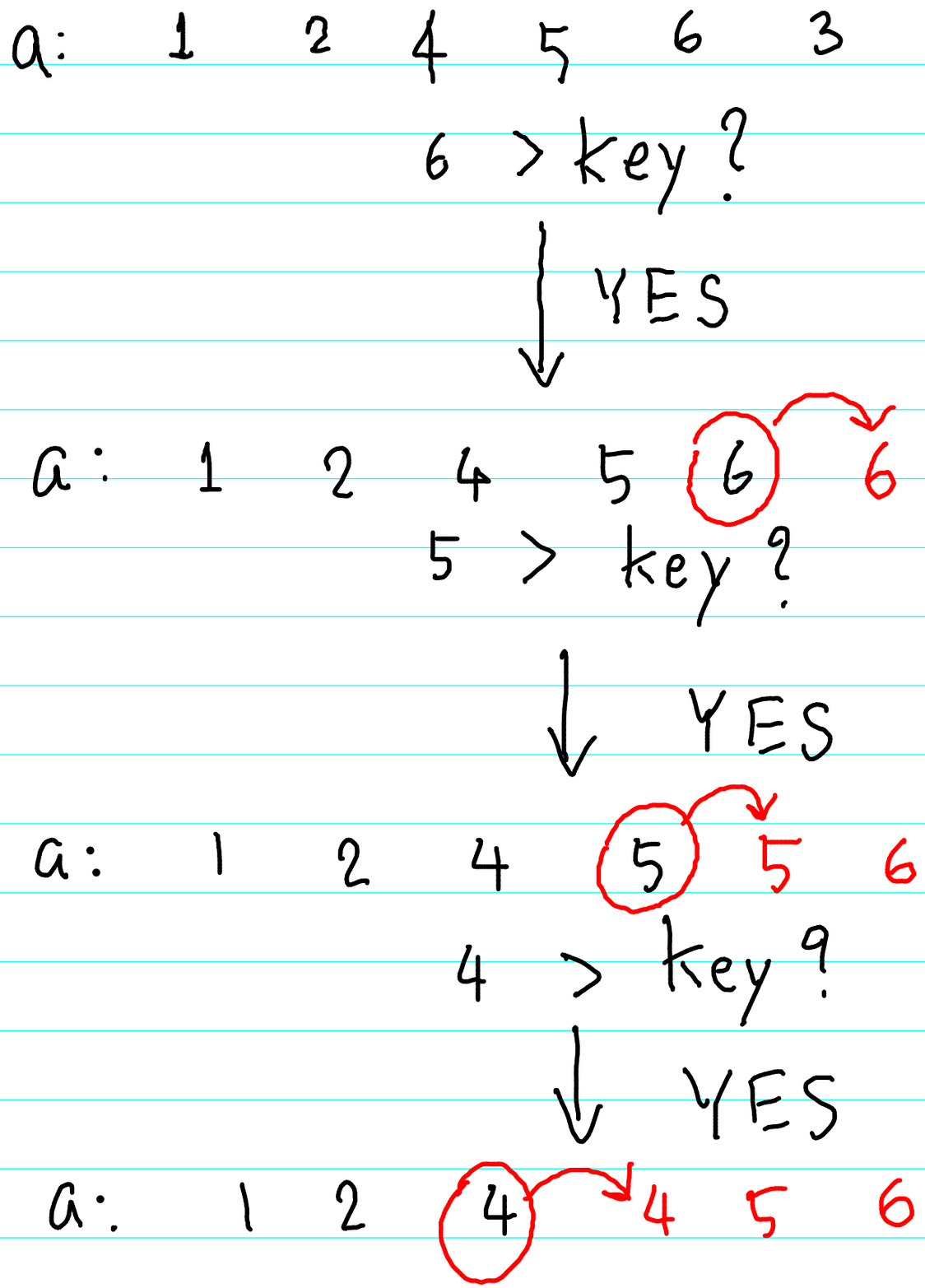
(1) เอาข้อมูลที่ต้องการแทรกเก็บไว้ในตัวแปร key

a: | 1 | 2 | 4 | 5 | 6 | 3

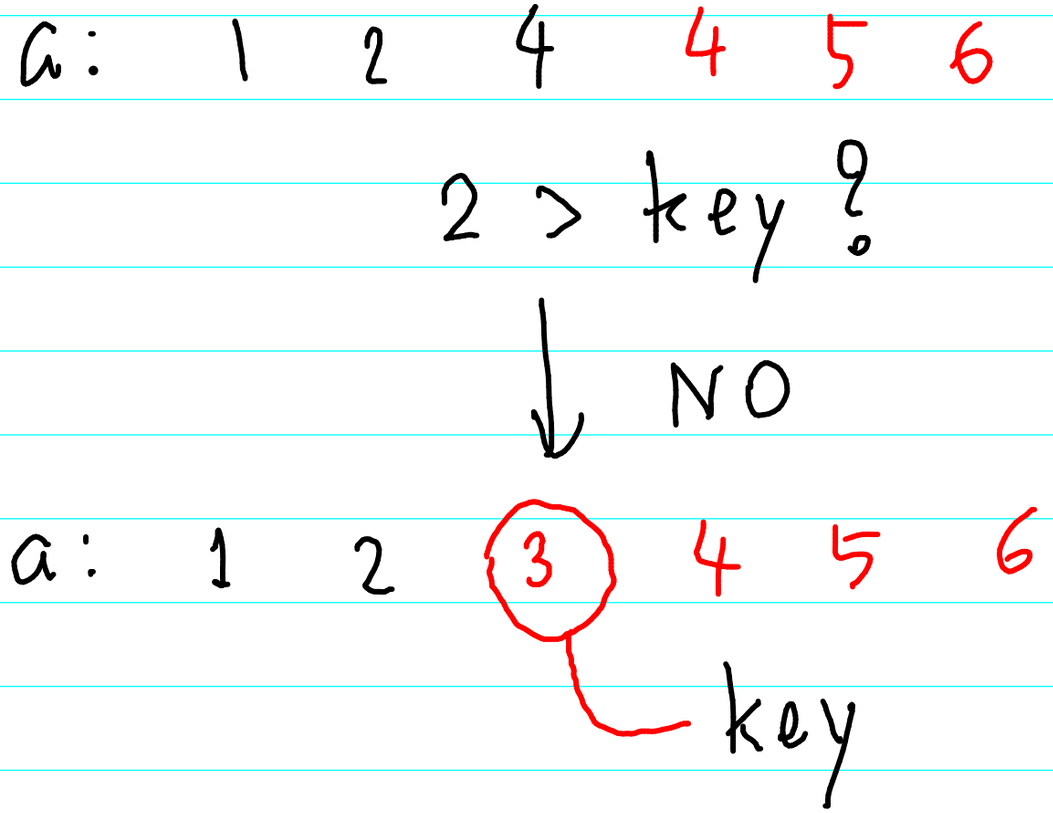
key = 3

(2) เปรียบเทียบ key กับค่าในสมาชิกเรียงลำดับจากซ้ายไปขวา

ถ้า key น้อยกว่าให้เลื่อนค่าของสมาชิกเรียงลำดับไปทางขวา


a: 1 2 4 5 6 3
6 > key ?
YES
a: 1 2 4 5 6 6
5 > key ?
YES
a: 1 2 4 5 5 6
4 > key ?
YES
a: 1 2 4 4 5 6

ถ้า key ไม่น้อยกว่าค่าในส่วน
ที่เรียงแล้วก็ให้ copy ค่า key
ใส่ลงในช่องที่อยู่ข้างขวาของค่านั้น



Pseudocode for Insertion Sort

```
INSERTION-SORT(a, n)
{
    for j ← 1 to n-1 do
        key ← a[j]
        i ← j-1
        while i > 0 and a[i] > key
            a[i+1] ← a[i]
            i ← i-1
        a[i+1] ← key
}
```

- เราจะวิเคราะห์การทำงานของ Insertion Sort

```
INSERTION-SORT(a, n)
{
for j ← 1 to n-1 do                          n-1
    key ← a[j]                               1/รอบ
    i ← j-1                                  1/รอบ
    while i > 0 and a[i] > key               ≤ j-1/รอบ   n-1 รอบ
        a[i+1] ← a[i]                        ≤ j-1/รอบ
        i ← i-1                              ≤ j-1/รอบ
    a[i+1] ← key                             1/รอบ
}
```

- loop ในสุดแต่ละรอบจะมีการทำงานมากที่สุดคือ

$$1 + 1 + 3(j-1) + 1 = 3j$$

- ดังนั้นเวลาการทำงาน worst case

$$T(n) = \sum_{j=1}^{n-1} 3j + (n-1)$$

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

$$= \frac{3n^2 - n - 2}{2}$$

- ถาม: เวลาการทำงาน worst case นี้เกิดขึ้นเมื่อไหร่?

- กรณี: เวลาการทำงาน best case ของ insertion sort มีค่าเท่าไหร่? และเกิดขึ้นเมื่อใด?

- Asymtotic Analysis

=> การวิเคราะห์อัลกอริทึมอย่างละเอียดขั้นที่ 2 ยังมีมากกว่าที่เราต้องการ

↳ เราได้ $T(n) = \frac{3n^2 - n - 2}{2}$

↳ เวลาเราบอก $T(n)$ คนอื่นเราคงจะพูดแค่ $T(n) \approx \frac{3n^2}{2}$

⇒ แต่สมประสิทธิ์ $\frac{3}{2}$ ก็ยังเป็นรายละเอียด
ที่มากเกินความจำเป็น

↳ เพราะมันขึ้นกับบันทอนจากจิกใน
การวิเคราะห์อัลกอริทึม

↳ ผมนับการเช็คว่า

$i > 0$ และ $a[i] > key$

เป็นคำสั่งเดียว

↳ ถ้านับเป็น 2 คำสั่งสมประสิทธิ์
จะเปลี่ยนเป็น $T(n) \approx 2n^2$

↳ แต่ไม่ว่าจะนับเป็นคำสั่งเดียวหรือ
2 คำสั่ง $T(n)$ ก็ยังอยู่ใน
รูป an^2 อยู่ดี

⇒ สังเกตว่า ข้อมูลที่สำคัญที่สุดของ $T(n)$ คือ ตัวที่มันอยู่ในรูป an^2 เท่านั้น สัมประสิทธิ์ของ n^2 ไม่สำคัญ

⇒ เมื่อนักคอมพิวเตอร์กล่าวถึงเวลาที่อัลกอริทึมใช้ทำงาน เขาจะตัดเทอมของ $T(n)$ ที่เล็กๆ ออกไป เหลือแต่เทอมที่ใหญ่ที่สุด นอกจากนี้จะตัดสัมประสิทธิ์ของเทอมนั้นออกไปอีก และจะใช้สัญกรณ์เชิงเส้นกำกับ (Asymptotic Notation) ในการบรรยายเวลาการเติบโต

$\Rightarrow$ เช่น เวลาในการทำงาน ของ insertion sort

คือ $T(n) = \frac{3n^2 - n - 2}{2}$

เราจะเขียนว่า $T(n) = O(n^2)$

$\Rightarrow$ เช่น เวลาในการทำงานของอัลกอริทึม

ที่ใช้แก้ปัญหาการหาผลบวกที่สูงสุด คือ

$T(n) = 3n - 1$

เราจะเขียนว่า $T(n) = O(n)$

⇒ การสนใจแค่เทอมที่ใหญ่ที่สุดของ $T(n)$ เท่ากับว่าเราสนใจ "อัตราการเจริญเติบโต" ของ $T(n)$ แทนที่จะสนใจเวลาการทำงานที่แท้จริงของมัน

⇒ สมมติว่าคุณมีอัลกอริทึม 2 อัลกอริทึมที่แก้ปัญหาเดียวกันได้โดย

* เวลาการทำงานของอัลกอริทึม 1.

$$T_1(n) = O(n^2)$$

* เวลาการทำงานของอัลกอริทึม 2

$$T_2(n) = O(n)$$

$\Rightarrow$ ดังนี้

$T_1(n)$ ลดลงเท่ากับ $n^2 - 3n + 6$

$T_2(n)$ ลดลงเท่ากับ $100n$

แน่นอนว่า n มีค่าน้อย ๆ

อัลกอริทึม 2 จะทำงานได้ช้ากว่าอัลกอริทึม 1

แต่เมื่อ n มีค่ามาก ๆ แล้วเราจะเห็นได้

ชัดว่าอัลกอริทึม 1 ทำงานได้ช้ากว่า

อัลกอริทึม 2 แน่นอน

$\Rightarrow$ นี่เป็นเพราะว่า $T_1(n)$ โตเร็วกว่า $T_2(n)$

กล่าวคือเมื่อ n มีค่ามาก ๆ $T_1(n) > T_2(n)$

⇒ การใช้ asymtotic notation ช่วยให้เราเปรียบเทียบเวลาการทำงานของอัลกอริทึมเมื่อ n มีค่ามากๆ ได้โดยไม่ต้องคำนึงถึงรายละเอียดในการวิเคราะห์

⇒ นอกจากนี้การวิเคราะห์อัลกอริทึมโดยมีเป้าหมายจะประมาณ $T(n)$ แทนที่จะหาค่า $T(n)$ โดยละเอียดทำให้เราวิเคราะห์อัลกอริทึมได้ง่ายขึ้น

- อัลกอริทึมต่อไปนี้ใช้เวลาทำงานเท่าไร?

```
for i ← 1 to n do
    for j ← 1 to i do
        for k ← 1 to j do
            x ← x + 1
```

```
for k ← 1 to j do
    x ← x + 1  ← 1 รอบ
```

} j รอบ

$\therefore O(j)$

```
for j ← 1 to i do
    for k ← 1 to j do
        x ← x+1
```

$O(j)$ ทำงาน } i รอบ

$$\therefore \sum_{j=1}^{i} O(j) = O(i^2)$$

```
for i ← 1 to n do
    for j ← 1 to i do
        for k ← 1 to j do
            x ← x + 1
```

$O(i^2)$ ทำงาน } n รอบ

$$\therefore \sum_{i=1}^{n} O(i^2) = O(n^3)$$