แผนการสอนสัปดาห์ที่ 3

บทที่ 2 การวิเคราะห์ความต้องการเชิงเวลาและเนื้อที่ๆ ต้องการของขั้นตอนวิธี จำนวนชั่วโมง 3

2.1 ประสิทธิภาพเชิงเวลา (Time Efficiency)

2.2 สัญลักษณ์เชิงเส้นกำกับทางเวลา Big O

จุดประสงค์การสอน (จุดประสงค์ทั่วไป

2.1 เข้าใจประสิทธิภาพเชิงเวลา (Time Efficiency)

2.2 รู้ถึงสัญลักษณ์เชิงเส้นกำกับทางเวลา Big O

ผลการเรียนรู้ (จุดประสงค์เฉพาะ

2.1 อธิบายประสิทธิภาพเชิงเวลา (Time Efficiency)

2.2 บอกถึงสัญลักษณ์เชิงเส้นกำกับทางเวลา Big O

วิธีสอนและกิจกรรมการเรียนการสอน

1. ผู้สอนแจ้งวัตถุประสงค์การเรียนรู้ให้นักศึกษาทราบ
2. ผู้สอนบรรยายนำเข้าสู่บทเรียน เนื้อหาในบทเรียน ใช้ power point ประกอบการบรรยายและเขียนอธิบายเพิ่มเติมบนกระดานไวท์บอร์ด
3. ให้ผู้เรียนตอบคำถามท้ายบทเรียน

สื่อการสอน/อุปกรณ์การสอน

1. กระดานไวท์บอร์ด
2. เอกสารประกอบการสอน
3. สไลด์ power point นำเสนอเนื้อหาประกอบการสอน

การวัดผล

1. สังเกตพฤติกรรมการเรียน
2. สอบกลางภาค

บทที่ 2 การวิเคราะห์ความต้องการเชิงเวลาและเนื้อที่ๆ ต้องการของขั้นตอนวิธี

ก่อนเข้าสู่หัวข้อ 2.1 และ 2.2 ผู้สอนทบทวนคณิตศาสตร์พื้นฐานที่จำเป็นในการนำไปใช้วัดประสิทธิภาพและความซับซ้อนของของโปรแกรมก่อนดังในสไลด์ต่อไปนี้

คณิตศาสตร์พื้นฐานเพื่อการวิเคราะห์ (Mathematic Preliminaries for Analysis)

ฟังก์ชั่นความซับซ้อนด้านเวลาของ algorithm:

Discrete Mathematic ที่มีการใช้งานบ่อย:

- ฟังก์ชั่น Exponential
- ฟังก์ชั่น Logarithm
- ฟังก์ชั่น Factorial

ฟังก์ชั่น Exponential

ฟังก์ชั่น Exponential อาศัยพื้นฐานทฤษฎีที่เกี่ยวข้องกับ**เลขยกกำลัง**

ถ้า a, b เป็นจำนวนจริง โดยที่ $a \neq 0$, $b \neq 0$ และ m, n เป็นจำนวนเต็ม

1. $a^m a^n = a^{m+n}$
2. $(a^m)^n = a^{mn}$
3. $(ab)^m = a^m b^m$
4. $(a/b)^m = a^m/b^m$
5. $a^m/a^n = a^{m-n}$

ฟังก์ชั่น Logarithms/ทฤษฎีที่เกี่ยวข้องกับฟังก์ชั่น Logarithms

→ฟังก์ชั่น Logarithm เป็นส่วนกลับกันของ ฟังก์ชั่น Exponential

กำหนด M, N, a และ b เป็นจำนวนจริงบวก โดยที่ $a \neq 1$, $b \neq 1$และมี n เป็นจำนวนจริง แล้ว สามารถสรุปได้ ดังนี้

1. $\log_a MN = \log_a M + \log_a N$
2. $\log_a M/N = \log_a M - \log_a N$
3. $\log_a 1 = 0$
4. $\log_a M^n = n\log_a M$
5. $\log_a a = 1$
6. $\log_a M = \log_a N$ ก็ต่อเมื่อ $M=N$
7. $a \log_a M = M$
8. $\log_{a^n} M = 1/n \log_a M$
9. $\log_{1/a} M = -\log_a M$
10. $\log_a M = 1/\log_M a$ เมื่อ $M \neq 1$

การวัดประสิทธิภาพของโปรแกรมสามารถ

พิจารณาจากการใช้เนื้อที่ในหน่วยความจำ (Space/Memory)

- ประสิทธิภาพของเวลาในการทำงาน (Time)
- การวัดเวลาทำการเปรียบเทียบได้ลำบาก
- พิจารณาอัตราการเติบโตของฟังก์ชัน (Growth Rates)

2.1 ประสิทธิภาพเชิงเวลา (Time Efficiency)

เวลาในการประมวลผลของโปรแกรมได้แก่

- Compile Time คือ เวลาที่ใช้ในการตรวจสอบไวยากรณ์ (syntax) ของ code ว่าเขียนได้ถูกต้องหรือไม่
- Run Time หรือ Execution Time คือ เวลาที่เครื่องทำการรันโปรแกรม
- คอมพิวเตอร์ใช้ในการประมวลผลลัพธ์

ประสิทธิภาพของโปรแกรม/อัลกอริธึ่มพิจารณาจากการวิเคราะห์ Space Complexity ได้ โดยดูจาก Maximum memory ว่ามีเท่าไหร่ที่รัน algorithm แล้ว มีจริงๆ เท่าไหร่

- วิเคราะห์ว่าต้องใช้หน่วยความจำทั้งหมดเท่าไหร่ในการประมวลผลอัลกอริธึ่มนั้น รองรับจำนวนข้อมูลที่ส่งเข้ามาประมวลผล ได้มากที่สุดเท่าใด เพื่อให้อัลกอริธึ่มนั้นสามารถประมวลผลได้อยู่
- ทราบขนาดของหน่วยความจำที่จะต้องใช้ประมวลผลอัลกอริธึ่มโดยไม่กระทบการประมวลผลอื่นๆ
- เพื่อเลือกคุณลักษณะของคอมพิวเตอร์ที่จะใช้ติดตั้งโปรแกรมที่พัฒนาขึ้นได้อย่างเหมาะสม

โดยองค์ประกอบของ Space Complexity ได้แก่

- Instruction Space

-จำนวนของหน่วยความจำที่คอมไพเลอร์จำเป็นต้องใช้ขณะทำการคอมไพล์โปรแกรม

- Data Space

-จำนวนหน่วยความจำที่ต้องใช้สำหรับเก็บค่าคงที่ และตัวแปรทั้งหมดที่ต้องใช้ในการประมวลผลโปรแกรม อันได้แก่ **Static memory allocation** ที่จำนวนของหน่วยความจำที่ต้องใช้อย่างแน่นอน ไม่มีการเปลี่ยนแปลง ประกอบด้วยหน่วยความจำที่ใช้เก็บค่าคงที่และตัวแปรประเภท array เช่น การประกาศตัวแปร

```
int    a, b;
char   s[10],   c;
```

นอกจากนี้ Dynamic memory allocation ที่จำนวนของหน่วยความจำที่ใช้ในการประมวลผลสามารถเปลี่ยนแปลงได้ และจะทราบจำนวนหน่วยความจำที่จะใช้ก็ต่อเมื่อโปรแกรมกำลังทำงานอยู่ เช่น การใช้ pointer และมีการจองเนื้อที่ใหน่วยความจำด้วยคำสั่ง malloc();

```
int  *p;
p = malloc(sizeof(int)*2);
```

2.2 สัญลักษณ์เชิงเส้นกำกับทางเวลา Big O

ประสิทธิภาพของอัลกอริธึ่ม จะพิจารณาจำนวนอิลิเมนต์ (Elements) ที่จะประมวลผลหรือจากจำนวนรอบการทำงานของตัวดำเนินการนั้นๆ

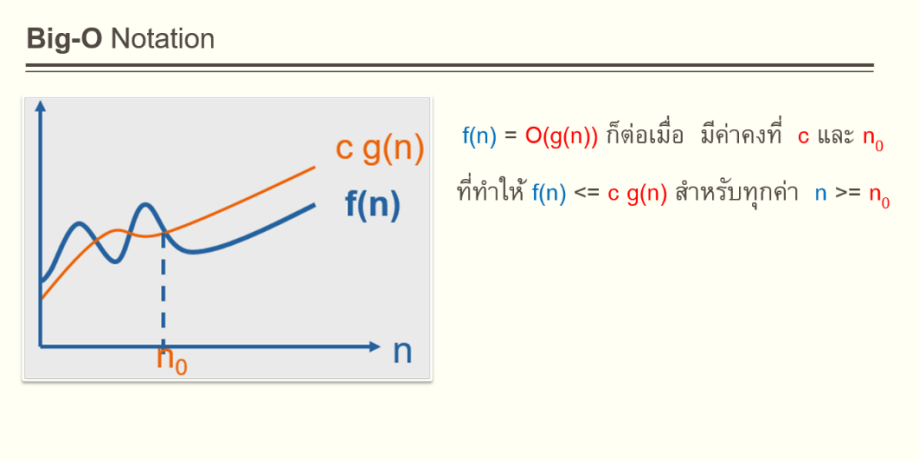
$$f(n) = efficiency$$

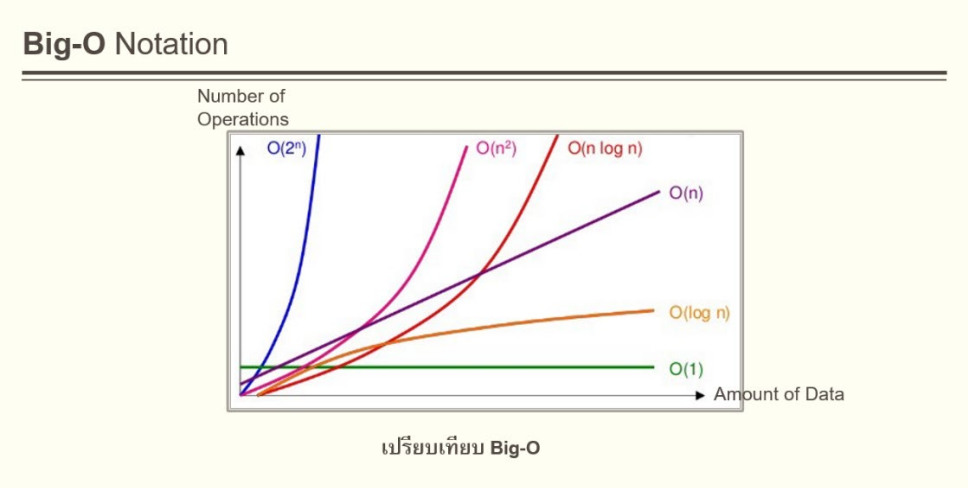
นั่นคืออัตราการเติบโตของฟังก์ชั่น (growth rates) ที่บอกความสัมพันธ์ระหว่างจำนวนข้อมูลนำเข้ากับความเร็วในการประมวลผลที่พิจารณาจากส่วนการทำงานของวนรอบประมวลผล (loop) เป็นสำคัญ

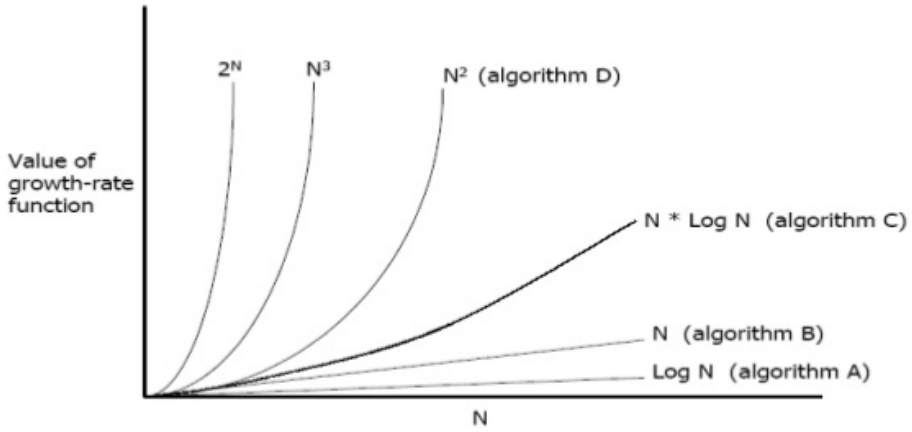
Asymptotic Notations (สัญลักษณ์อะซิมพ์ทอทิค)

เป็นสัญลักษณ์ที่ใช้นำเสนอความซับซ้อนด้านเวลาของอัลกอริธึ่ม มีหลายแบบสัญลักษณ์ ได้แก่

- Big-O notation คือ Asymptotic upper bound ที่บอกขอบเขตบนของฟังก์ชั่น Worst-case โดยจะพิจารณาปริมาณอินพุทข้อมูลมากๆ ซึ่งการใช้สัญลักษณ์ Big-O มีวัตถุประสงค์เพื่อ
  - อธิบายขอบเขต(บน) ของฟังก์ชันการเติบโตทางด้านเวลา
  - ใช้พิจารณาความซับซ้อนด้านเวลาของฟังก์ชั่น
  - เพื่อเลือกอัลกอริธึ่มที่มีประสิทธิภาพสูงสุด (ไม่ได้มีวัตถุประสงค์เพื่อวัดเวลา)
- Big-Ω (Big-Omega) notation คือ Asymptotic lower bound ที่บอกขอบเขตล่างของฟังก์ชั่น best case
- Big-θ (Big-Theta) notation คือ Asymptotic tight bound ที่บอกขอบเขตในช่วงกลางของฟังก์ชั่น average case
- Little-o notation คือ Asymptotic bound ของฟังก์ชั่น





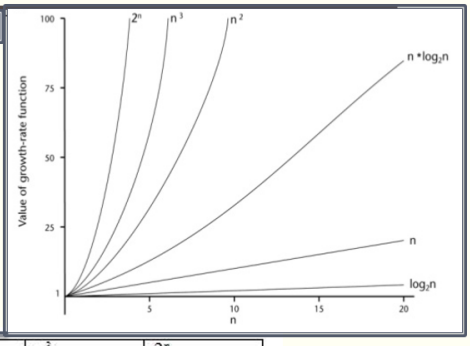


ภาพที่ 3.1 เปรียบเทียบประสิทธิภาพของอัลกอริธึ่มต่างๆ

ในการแก้ปัญหาทางคอมพิวเตอร์ เราอาจออกแบบอัลกอริธึ่มไว้หลายอัลกอริธึ่ม เช่น 4 อัลกอริธึ่ม (A-D) ภาพที่ 3.1 จะเห็นว่า อัลกอริธึ่ม A ถึง D มีประประสิทธิภาพสูงสุดเรียงลำดับจากมากไปน้อย เมื่อพิจารณาจากอัตราการเติบโตของฟังก์ชั้นที่มีความคงตัวมากไปยังน้อยที่สุด

Functions of Growth Rate (อัตราการเติบโตของฟังก์ชั่น) เรียงจากน้อยไปมาก

Function	Name	Algorithms
c	Constant	Array index lookup
$\log N$	Logarithmic	Binary search
$\log^2 N$	Log-squared	
N	Linear	Graph traversal
$N \log N$	N log N	Merge, Quick sort
N^2	Quadratic	Shortest path
N^3	Cubic	Dynamic programming
2^N	Exponential	Traveling salesman



g(n) / n	log n	n	n log$_2$ n	n^2	n^3	2^n
5	3	5	15	25	125	32
10	4	10	40	100	10^3	10^3
100	7	100	700	10^4	10^6	10^{30}
1000	10	10^3	10^4	10^6	10^9	10^{300}

[ภาพ: internet 2017]

อ. ชิดาวรร คล้ายศรี มทรพ. วิทยาการคอมพิวเตอร์ 10

ภาพที่ 3.2 เปรียบเทียบอัตราการเติบโตของฟังก์ชั่นของอัลกอริธึ่มต่างๆ

แผนการสอนสัปดาห์ที่ 4

บทที่ 2 การวิเคราะห์ความต้องการเชิงเวลาฯ (ต่อ) **จำนวนชั่วโมง 3**

2.3 ประมาณความต้องการเชิงเวลา (Time Estimation)

2.4 ประสิทธิภาพเชิงเนื้อที่ (Space Efficiency)

2.5 การวิเคราะห์ชั้นความซับซ้อน

จุดประสงค์การสอน (จุดประสงค์ทั่วไป

2.3 รู้ถึงการประมาณความต้องการเชิงเวลา (Time Estimation)

2.4 รู้ถึงประสิทธิภาพเชิงเนื้อที่ (Space Efficiency)

2.5 เข้าใจการวิเคราะห์ชั้นความซับซ้อน

ผลการเรียนรู้ (จุดประสงค์เฉพาะ

2.3 บอกประมาณความต้องการเชิงเวลา (Time Estimation)

2.4 บอกประสิทธิภาพเชิงเนื้อที่ (Space Efficiency)

2.5 อธิบายการวิเคราะห์ชั้นความซับซ้อน

วิธีสอนและกิจกรรมการเรียนการสอน

1. ผู้สอนแจ้งวัตถุประสงค์การเรียนรู้ให้นักศึกษาทราบ

2. ผู้สอนบรรยายนำเข้าสู่บทเรียน เนื้อหาในบทเรียน ใช้ power point ประกอบการบรรยายและเขียนอธิบายเพิ่มเติมบนกระดานไวท์บอร์ด

3. ให้ผู้เรียนตอบคำถามท้ายบทเรียน

สื่อการสอน/อุปกรณ์การสอน

1. กระดานไวท์บอร์ด
2. เอกสารประกอบการสอน
3. สไลด์ power point นำเสนอเนื้อหาประกอบการสอน

การวัดผล

1. สังเกตพฤติกรรมการเรียน
2. สอบกลางภาค

(ต่อ) บทที่ 2 การวิเคราะห์ความต้องการเชิงเวลาและเนื้อที่ๆ ต้องการของขั้นตอนวิธี

2.3 ประมาณความต้องการเชิงเวลา (Time Estimation)

การวิเคราะห์ Time Complexity คือ เวลาที่เครื่องคอมพิวเตอร์ต้องใช้ในการประมวลผลอัลกอริธึ่ม ซึ่งวิเคราะห์เพื่อ:

- ประมาณการเวลาทั้งหมดที่ต้องใช้ในโปรแกรมได้
- มุ่งแก้ไขไปที่อัลกอริธึ่มที่ใช้เวลาในการประมวลผลนานๆ ทำให้ไม่ต้องแก้ไขทั้งโปรแกรม
- เลือกคุณลักษณะของคอมพิวเตอร์ที่จะใช้ติดตั้งโปรแกรมที่พัฒนาขึ้นได้อย่างเหมาะสม

2.4 ประสิทธิภาพเชิงเนื้อที่ (Space Efficiency)

ประสิทธิภาพกับการจัดเก็บข้อมูล

จาก ตัวอย่างที่ผ่านมาเวลาที่ใช้ทำงานจะขึ้นกับขนาดหรือจำนวนของข้อมูลที่รับ เข้ามาในปัญหาอื่นๆอาจขึ้นกับวิธีจัดการกับข้อมูลที่รับเข้ามา เช่นการจดเรียงลำดับข้อมูลจะใช้เวลาต้องใช้เวลาจัดเรียงลำดับมากขึ้น

ดัง นั้นในการพยายามวัดผลเวลาของ T อาจใช้ในกรณีดีที่สุด (Best Case) ในกรณีเฉลี่ย (Aver age Case) หรือในกรณีแย่ที่สุด (Worst Cast) การวัดสิทธิภาพของอัลกอริทึมในกรณีดีที่สุดดูเป็นเรื่องง่ายแต่ไม่ สามารถนำมาวัดผลได้และในกรณีเฉลี่ยก็เป็นการยากที่จะต้องพิจารณาค่าเฉลี่ย ที่ชัดเจนแต่ในกรณีที่แย่ที่สุดเหมือนไม่ง่ายแต่ก็ไม่ยากที่จะนำมาใช้วัดผล ดังนั้นเวลา T(n) จึงนำมาใช้วัดประสิทธิภาพของอัลกอริทึมในกรณีที่แย่ที่สุด

2.5 การวิเคราะห์ขั้นความซับซ้อน

Analysis of Algorithms (การวิเคราะห์อัลกอริธึ่ม

ในการวิเคราะห์ประสิทธิภาพของอัลกอริธึ่มทางคอมพิวเตอร์ที่ได้ออกแบบไว้นั้น เรามักพิจารณาจากความซับซ้อนของอัลกอริธึ่มใน 2 ด้านคือ

- Time Complexity (ความซับซ้อนด้านเวลา)

วัดจากจำนวน n รอบที่ใช้ในการคำนวณ (+ - * /) เปรียบเทียบทางตรรกศาสตร์ (> = <) ใน CPU พิจารณาจาก loops

- Space Complexity (ความซับซ้อนด้านเนื้อที่)

วัดจากเนื้อที่ทั้งหมดที่ใช้ในการเก็บข้อมูล ตัวแปรต่างๆ ไว้ใน main memory แต่จะมีความสำคัญน้อยกว่าความซับซ้อนด้านเวลา

ลูปแบบลอการิธมิค (Logarithmic Loops)

= เพิ่มหรือลดค่าภายในลูปสองเท่า

- Multiply Loops

```
for( i =1; i <= 1000; i* = 2 )
    application code
```

- Divide Loops

```
for( i =1000; i  >= 1; i / = 2 )
    application code
```

ประสิทธิภาพของอัลกอริทึม $f(n) = [\log_n]$ หรือ $[\log_2 n]$

ลูปแบบเชิงเส้น (Linear Loops)

= มีการเพิ่ม หรือลดค่าภายในลูปแบบคงที่

```
for ( i = 0; i < 1000; i++ )
    application code
```

ประสิทธิภาพของอัลกอริธึ่มคือ $f(n) = n$

```
for ( i = 0; i < 1000; i+= 2 )
    application code
```

ประสิทธิภาพของอัลกอริธึ่ม $f(n) = n / 2$

→เมื่อพล้อตจุดลงกราฟจะมีลักษณะเป็นเส้นตรงเหมือนกัน

ลูปแบบซ้อน (Nested Loops)

= ภายในลูป จะมีลูปซ้อนอีกลูปหนึ่ง

✓ ยอดรวมที่ได้คือผลคูณของจำนวนลูปชั้นใน (Inner Loop) กับจำนวนของลูปชั้นนอก (Outer Loop)

✓ ลูปแบบซ้อนชนิดลอการิธมิคเชิงเส้น (Linear Logarithmic)

- ลูปแบบซ้อนชนิดกำลังสอง (Quadratic)
- ลูปแบบซ้อนกำลังสองชนิดขึ้นต่อกัน (Dependent Quadratic)

ลูปแบบซ้อนชนิดกำลังสอง (Quadratic)

```
for (i = 0; i < 10; i++)
    for (j = 0; j < 10; j++)
        application code
```

จำนวนรอบคือ 10 * 10

$f(n) = n^2$

ลูปแบบซ้อนชนิดลอการิธมิคเชิงเส้น

```
for (i = 0; i < 10; i++)
    for (j = 1; j <= 10; j*= 2)
        application code
```

จำนวนรอบคือ 10 $\log_{10}$

$f(n) = n \log n$

จำนวนรอบการทำงานในลูปในคือ 1+2+3 ... + 10 = 55

f จำนวนรอบเฉลี่ยคือ 55/10 = 5.5

เทียบได้เท่ากับ **(n + 1)/2**

(n) = n * (n+1)/2