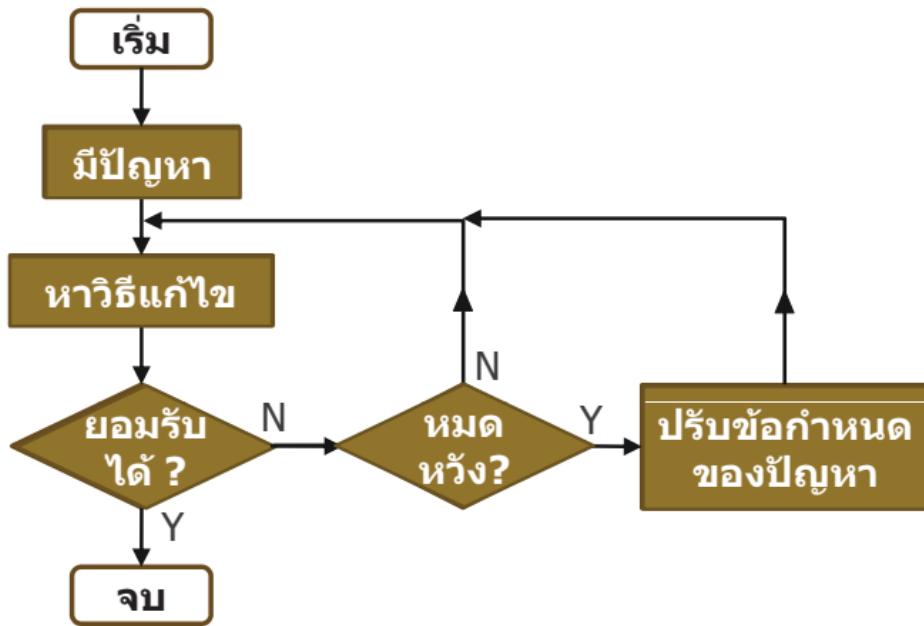


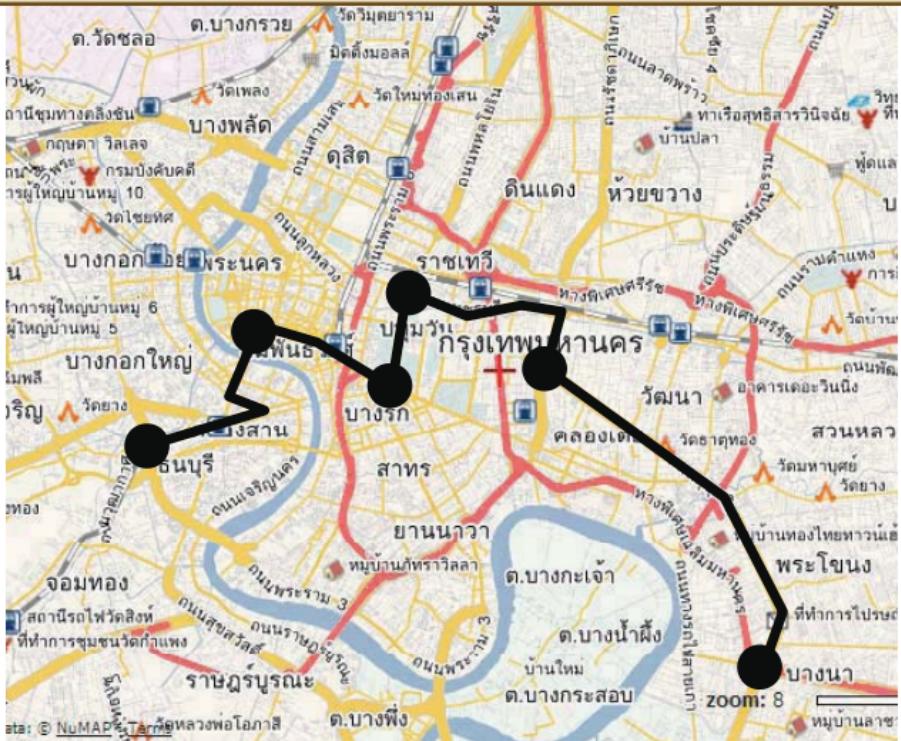
# การออกแบบอัลกอริทึม

สมชาย ประสิทธิ์จูตระกูต

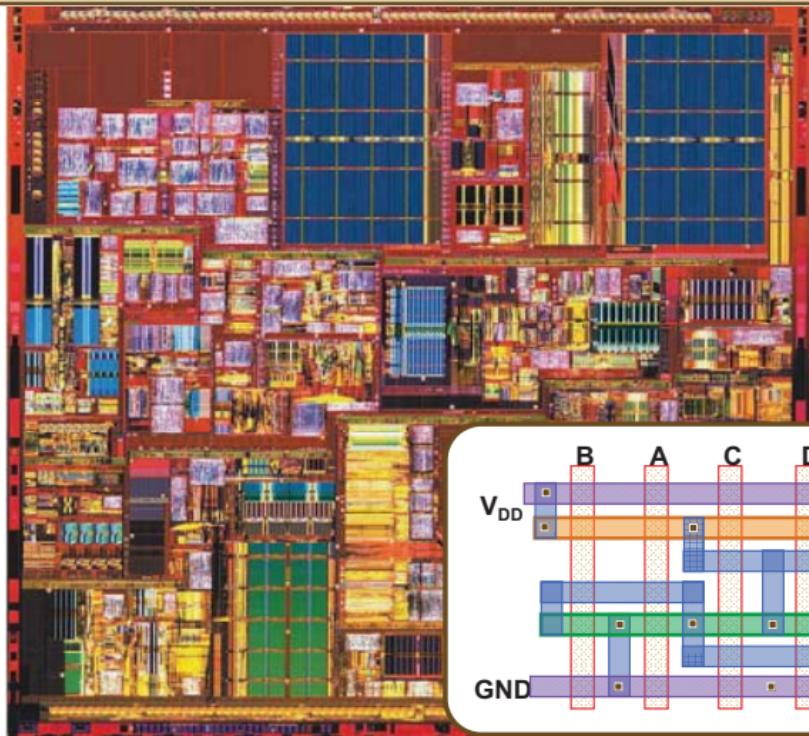
# วงจรการออกแบบอัลกอริทึม



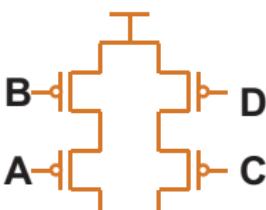
รถโรงเรียน



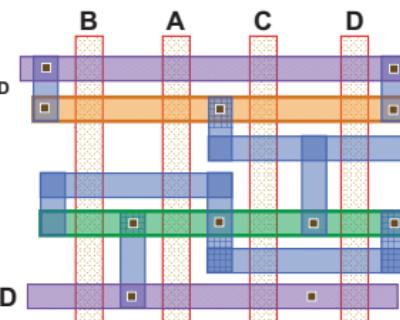
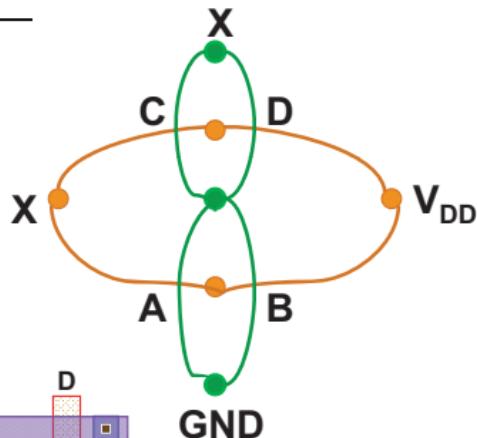
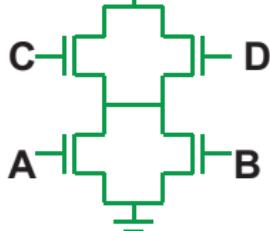
## การออกแบบวงจรรวม



# การออกแบบวงจร



$$X = \overline{(A+B) \cdot (C+D)}$$



# 数独

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

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

# ดาวมหาลัย : สามารถ

หนูดาวเป็นลูกสาวก  
จบชั้นม.6 โรงเรียนบ้านหนองใหญ่  
คุณแม่ขายไร์ขายนา ส่งดาวเข้ามาเรียนมหาลัย  
มาเรียนอยู่ในกรุงเทพ ยูนิเวอร์ซิตี้ที่หันสมัย  
ดาวสวยดาวเริดดาวเด่น คนมั่นสวยทำอะไรก็เด่น  
ก็เลยถูกพรีเซ็นเตอร์ให้เป็นดาวมหาลัย  
เพื่อนๆที่คุณะ ปลื้มดาวสุดๆเลยนะจะ

หนู, ดาว, เป็น, ลูก, สาว, กก,  
จบ, ชั้น, ม 6, โรงเรียน, บ้าน, หนอง, ในญี่,  
คุณ, แม่, ขาย, ไร์, ขาย, นา, ส่ง, ดาว, เข้า, มา, เรียน, มหาลัย,  
มา, เรียน, อยู่, ใน, กรุงเทพ, ยูนิเวอร์ซิตี้, ที่, หันสมัย,  
ดาว, สวย, ดาว, เริด, ดาว, เด่น, คน, มั่น, สวย, ทำ, อะไร, ก็, เด่น,  
ก็, เลย, ถูก, พรีเซ็นเตอร์, ให้, เป็น, ดาว, มหาลัย,  
เพื่อน, ๆ, ที่, คุณะ, ปลื้ม, ดาว, สุด, ๆ, เลย, จะ, จะ,

## การรู้จำ

สมชาย

Somchai



53 4F 4D 43 48 41 49



# ข้อกำหนดของปัญหา

- ❖ ลักษณะของข้อมูลขาเข้า (input)
- ❖ ผลลัพธ์ที่ต้องการ
- ❖ ต.ย. : ปัญหาตัวหมู่มาก (majority problem)

❖ input : ข้อมูลจำนวน  $n$  ตัว

❖ output : คืน TRUE เมื่อมีข้อมูลที่มีค่าซ้ำกันเกิน  $\lfloor n/2 \rfloor$   
: ไม่เช่นนั้น คืน FALSE

problem

problem instance  
input instance

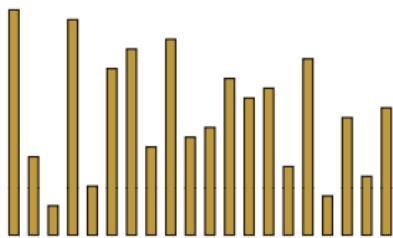
1	4	3	3	3	1	3
---	---	---	---	---	---	---

ตัวหมู่มากคือ 3  $\rightarrow$  TRUE

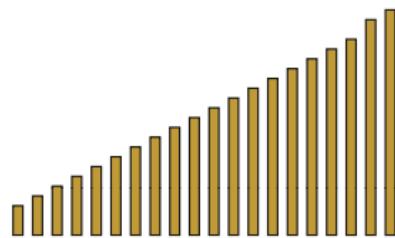
1	4	7	3	3	1	2
---	---	---	---	---	---	---

ไม่มีตัวหมู่มาก  $\rightarrow$  FALSE

# Sorting

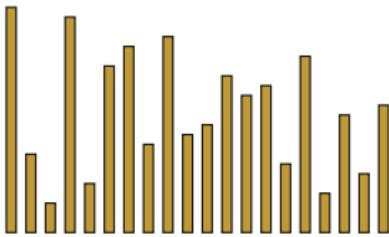


**Input**



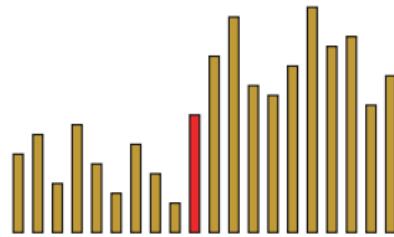
**Output**

# Median and Selection



**k = 10**

**Input**

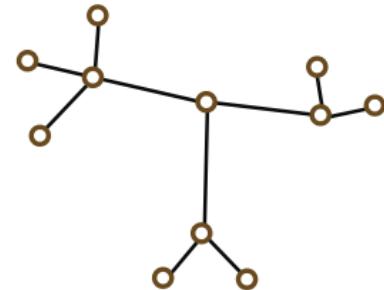


**Output**

# Minimum Spanning Tree

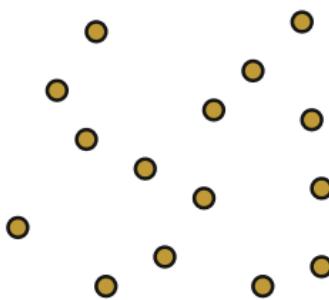


Input

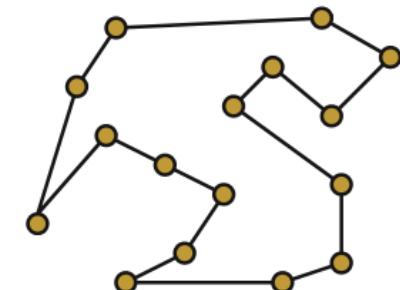


Output

# Traveling Salesman Problem

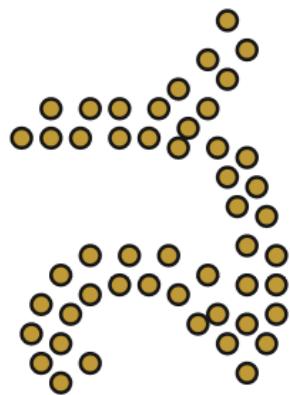


Input

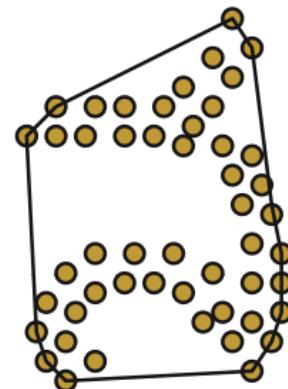


Output

# Convex Hull

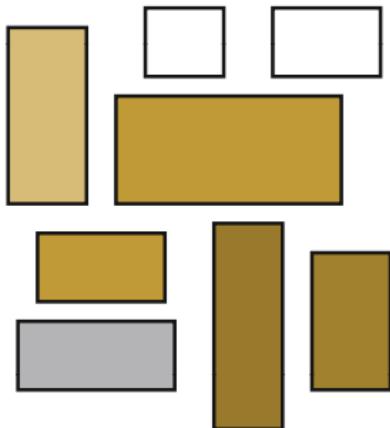


Input

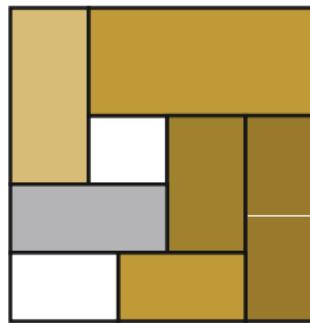


Output

# Bin Packing



Input



Output

# Primality Testing

8338169264555846052842102071

NO

Input

Output

# Factoring

8338169264555846052842102071

**Input**

$$\begin{array}{r} 179424673 \\ \times 2038074743 \\ \times 22801763489 \\ = \\ 8338169264555846052842102071 \end{array}$$

**Output**

# String Matching

You are the fairest of your sex,  
Let me be your hero;  
I love you as one over x,  
As x approaches zero.  
Positively.

you

Input

You are the fairest of **your** sex,  
Let me be **your** hero;  
**I** love **you** as one over x,  
As x approaches zero.  
Positively.

Output

# Approximate String Matching

You are the fairest of your sex,  
Let me be your hero;  
I love you as one over x,  
As x approaches zero.  
Positively.

heero

Input

You are the fairest of your sex,  
Let me be your hero;  
I love you as one over x,  
As x approaches zero.  
Positively.

Output

# Satisfiability

$$(x + y)(x + \bar{y})$$

$x = \text{true}$

**YES**

$$(x + y)(x + \bar{y})\bar{x}$$

**NO**

**Input**

**Output**

# Halting

x = 7

```
while (x > 1) {  
    if (x is even)  
        then x = x/2  
    else x = 3x+1  
}
```

x = 7

```
while (x ≥ 1) {  
    if (x is even)  
        then x = x/2  
    else x = 3x+1  
}
```

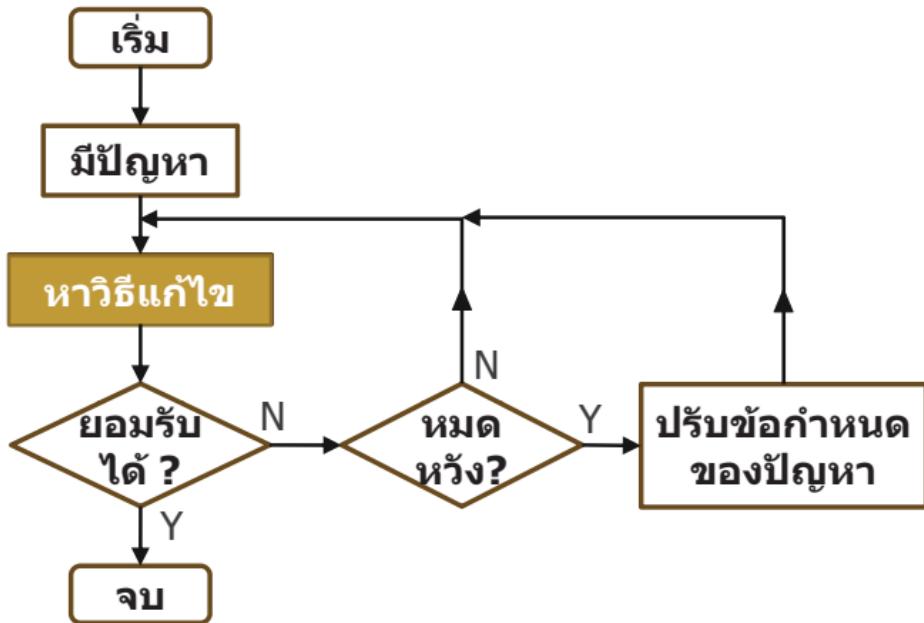
Input

YES

NO

Output

# ขั้นตอนการออกแบบอัลกอริทึม

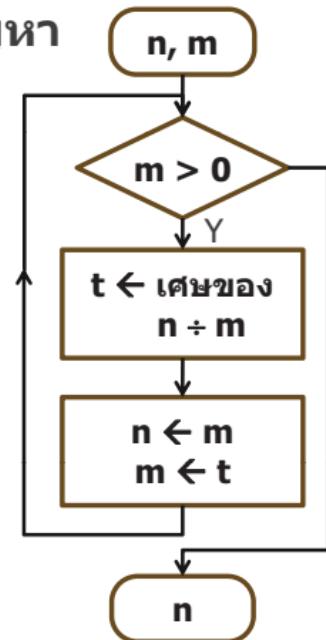


# อัลกอริทึม (Algorithm)

- ❖ ลำดับการทำงาน เชิงคำนวณซึ่ง  
แปลงตัวอย่างข้อมูลขาเข้าของปัญหา  
ไปเป็นผลลัพธ์ที่ต้องการ
- ❖ ขั้นตอนวิธีการแก้ไขปัญหาด้วย  
คอมพิวเตอร์

```
GCD( n, m ) {
    while ( m > 0 ) {
        t = n mod m
        n = m
        m = t
    }
    return n
}
```

Euclid  
Algorithm



## อัลกอริทึมการหารากที่ 3

- ❖ โดยใช้เครื่องคิดเลขถูก ๆ ที่มีปุ่มหารากที่สอง  $\frac{1}{x^2}$
- ❖ กดจำนวนที่ต้องการหารากที่ 3
- ❖ กดปุ่มหารากที่สอง 1 ครั้ง และกดปุ่มคูณ  $\xrightarrow{x^2} \frac{1}{2}\left(1+\frac{1}{2^2}\right)$
- ❖ กดปุ่มหารากที่สอง 2 ครั้ง และกดปุ่มคูณ  $\xrightarrow{x^2} \frac{1}{2}\left(1+\frac{1}{2^2}\right)\left(1+\frac{1}{2^4}\right)$
- ❖ กดปุ่มหารากที่สอง 4 ครั้ง และกดปุ่มคูณ  $\xrightarrow{x^2} \frac{1}{2}\left(1+\frac{1}{2^2}\right)\left(1+\frac{1}{2^4}\right)\left(1+\frac{1}{2^8}\right)$
- ❖ กดปุ่มหารากที่สอง 8 ครั้ง และกดปุ่มคูณ  $\xrightarrow{x^2} \frac{1}{2}\left(1+\frac{1}{2^2}\right)\left(1+\frac{1}{2^4}\right)\left(1+\frac{1}{2^8}\right)$
- ❖ ...
- ❖ ทำเช่นนี้จนกระทั่งเมื่อกดปุ่มรากที่สองแล้วได้ 1 เสมอ
- ❖ หลังกดปุ่มคูณครั้งสุดท้าย กดปุ่มรากที่สองอีกครั้ง จะได้คำตอบ

$$x^{\frac{1}{3}} = x^{\frac{1}{2^2}\left(1+\frac{1}{2^2}\right)\left(1+\frac{1}{2^4}\right)\left(1+\frac{1}{2^8}\right)\left(1+\frac{1}{2^{16}}\right)\dots}$$

# ตัวอย่างมาก : นับแต่ละตัวว่ามีกี่ตัว

1	4	3	3	3	1	3
---	---	---	---	---	---	---



1	4	7	3	3	1	2
---	---	---	---	---	---	---

```
hasMajority( d[1..n] ) {
    for (i = 1; i <= n; i++) {
        c = 0;
        for (j = 1; j <= n; j++)
            if (d[i] == d[j]) c++
        if (c > n/2) return TRUE
    }
    return FALSE
}
```

- |                |       |
|----------------|-------|
| นับ 1 มี 2 ตัว |       |
| นับ 4 มี 1 ตัว |       |
| นับ 3 มี 4 ตัว | true  |
| นับ 1 มี 2 ตัว |       |
| นับ 4 มี 1 ตัว |       |
| นับ 7 มี 1 ตัว |       |
| นับ 3 มี 2 ตัว |       |
| นับ 3 มี 2 ตัว |       |
| นับ 1 มี 2 ตัว |       |
| นับ 2 มี 1 ตัว | false |

## อัลกอริทึม

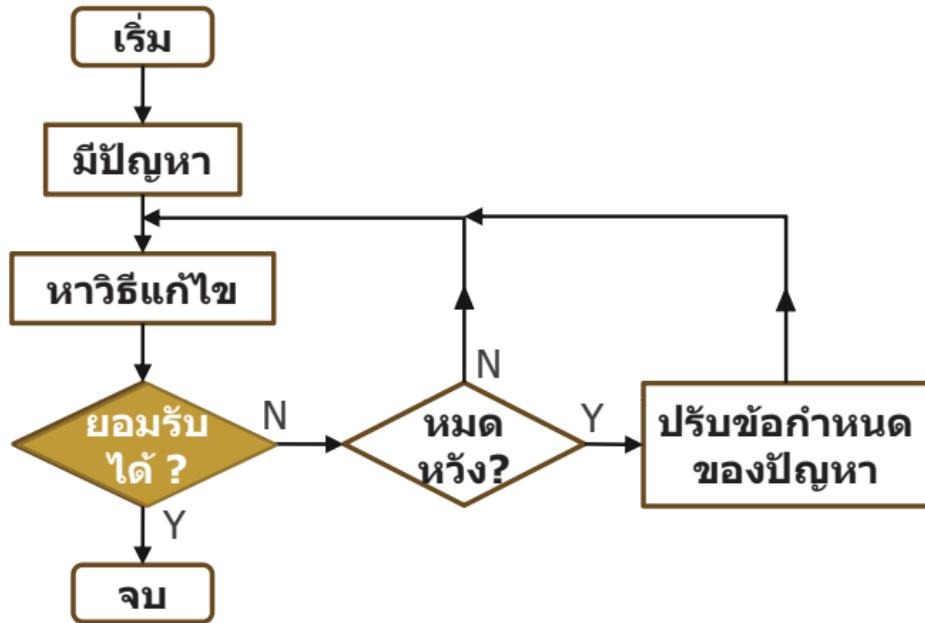
- ❖ ต้องหาผลลัพธ์ที่ถูกต้อง ในเวลาอันจำกัด ให้กับข้อมูลขาเข้าทกรูปแบบของปัญหา
- ❖ เราต้องการอัลกอริทึมที่
  - ❖ ทำงานถูกต้อง
  - ❖ ทำงานได้รวดเร็ว มีประสิทธิภาพ
  - ❖ ใช้หน่วยความจำน้อย ๆ
  - ❖ ทำความเข้าใจได้ง่าย

# กลวิธีการอุปกรณ์แบบอัลกอริทึม

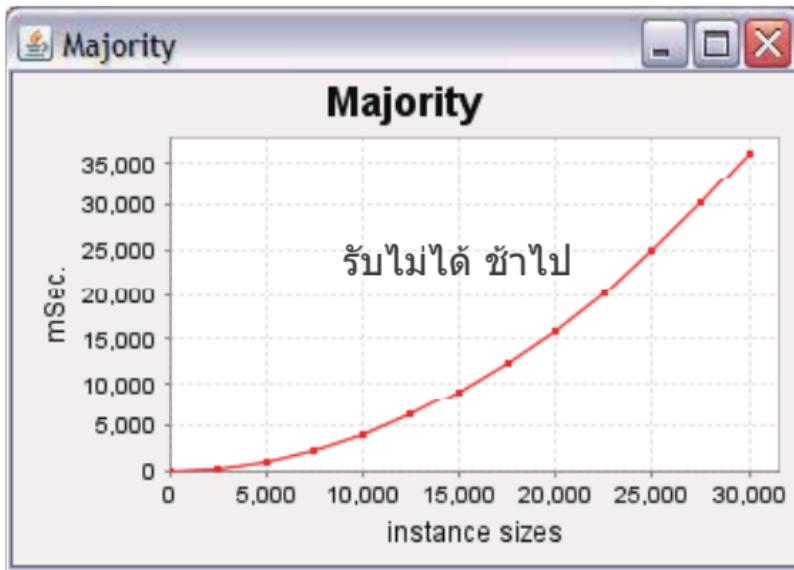
- Brute force**
- Divide and Conquer**
- Dynamic Programming**
- Greedy Algorithm**
- State-Space Search**
- Randomized Algorithm**
- Approximation Algorithm**

...

# ขั้นตอนการออกแบบอัลกอริทึม



# เวลาการทำงาน



กรณีอาร์เรย์ไม่มีตัวหมุนมาก

# ตัวหมู่มาก : sort + นับ

1	4	3	3	3	1	3
---	---	---	---	---	---	---

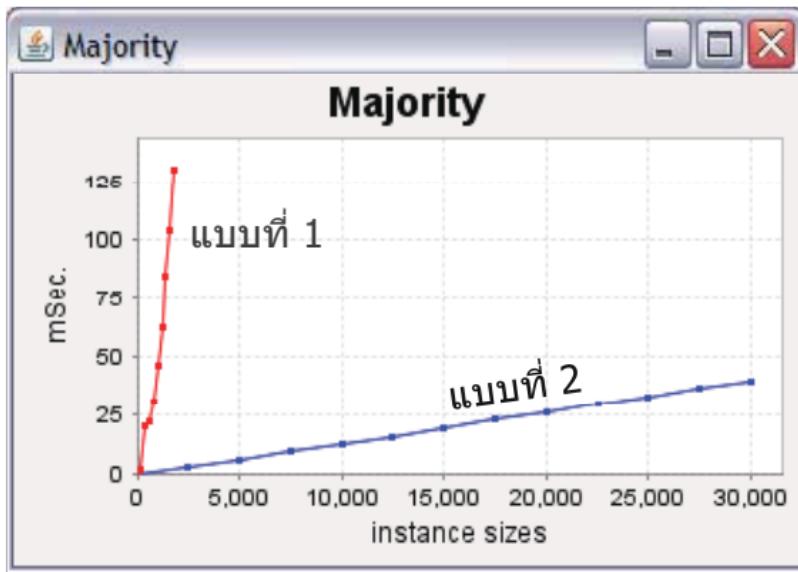
1	1	3	3	3	3	4
---	---	---	---	---	---	---



นับ 1 มี 2 ตัว  
นับ 3 มี 4 ตัว true

```
hasMajority( d[1..n] ) {  
    sort( d );  
    c = 0; i = 1  
    for (j = 1; j <= n; j++)  
        if (d[i] == d[j]) {  
            if (++c > n/2) return TRUE  
        } else {  
            i = j; c = 1  
        }  
    return FALSE  
}
```

# เวลาการทำงาน



กรณีอาเรย์ไม่มีตัวหมุนมาก

# ตัวหมู่มาก : sort + นับตัวกาง

1	1	3	3	3	4
1	1	3	3	3	4

นับ 3 มี 4 ตัว

1	1	2	3	3	4	7
1	1	2	3	3	4	7

นับ 3 มี 2 ตัว

```
hasMajority( d[1..n] ) {  
    sort( d );  
    c = 0; i = ⌈n/2⌉;  
    for (j = 1; j <= n; j++)  
        if (d[i] == d[j]) c++  
    if (c > n/2) return TRUE  
    return FALSE  
}
```

# ตัวหมู่มาก : sort + นับตัวกาง

1	1	3	3	3	3	4
---	---	---	---	---	---	---

นับ 3 มี 4 ตัว

1	1	2	3	3	4	7
---	---	---	---	---	---	---

นับ 3 มี 2 ตัว

```

hasMajority( d[1..n] ) {
    sort( d );
    c = 0; i = ⌈n/2⌉
    for (j = i; j>=1 AND d[i]==d[j]; j--) c++
    for (j = i+1; j<=n AND d[i]==d[j]; j++) c++
    if (c > n/2) return TRUE
    return FALSE
}

```

# ตัวหน่วยมาก : สุ่ม + นับ

1	4	7	3	3	1	2
---	---	---	---	---	---	---

สุ่มได้ 4 → นับ 4 → มี 1 ตัว 

1	4	3	3	3	1	3
---	---	---	---	---	---	---

สุ่มได้ 3 → นับ 3 → มี 4 ตัว 

1	4	3	3	3	1	3
---	---	---	---	---	---	---

สุ่มได้ 3 → นับ 3 → มี 4 ตัว 

1	4	3	3	3	1	3
---	---	---	---	---	---	---

สุ่มได้ 1 → นับ 1 → มี 2 ตัว 

# ตัวหมู่มาก : สุ่ม + นับ

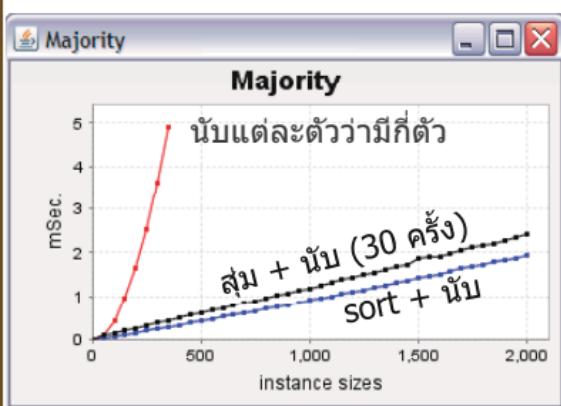
```
hasMajority( d[1..n] ) {
    c = 0; i = random(1, n)
    for (j = 1; j <= n; j++)
        if (d[i] == d[j]) c++
    if (c > n/2) return TRUE
    return FALSE
}
```

โอกาสที่ของจริงมีตัวหมู่มากแต่สุ่มไม่พบ  $< \frac{1}{2}$

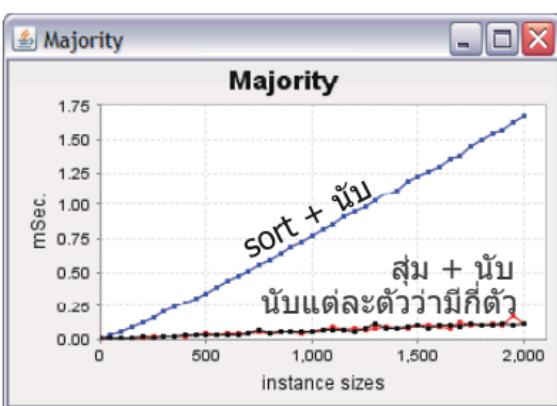
```
hasMajority( d[1..n] ) {
    for (k = 1; k<=30; k++) {
        c = 0; i = random(1, n)
        for (j = 1; j <= n; j++)
            if (d[i] == d[j]) c++
        if (c > n/2) return TRUE
    }
    return FALSE
}
```

โอกาสที่ของจริงมีตัวหมู่มากแต่สุ่มไม่พบ  $< (\frac{1}{2})^{30}$

# เปรียบเทียบเวลาการทำงาน



กรณีอาเรย์ไม่มีตัวหนู่มาก



กรณีอาเรย์มีตัวหนู่มาก

# การวิเคราะห์เวลาการทำงาน

## ❖ Experimental Analysis

- ❖ แปลงอัลกอริทึมเป็นโปรแกรม
- ❖ สั่งทำงานกับข้อมูลขนาดใหญ่ๆ แบบ
- ❖ จับเวลา วาดกราฟ หาความสัมพันธ์ของเวลา กับปริมาณข้อมูล

## ❖ Mathematical Analysis

- ❖ วิเคราะห์หาความสัมพันธ์ของจำนวนคำสั่งที่ทำงาน กับปริมาณข้อมูลที่ได้รับ

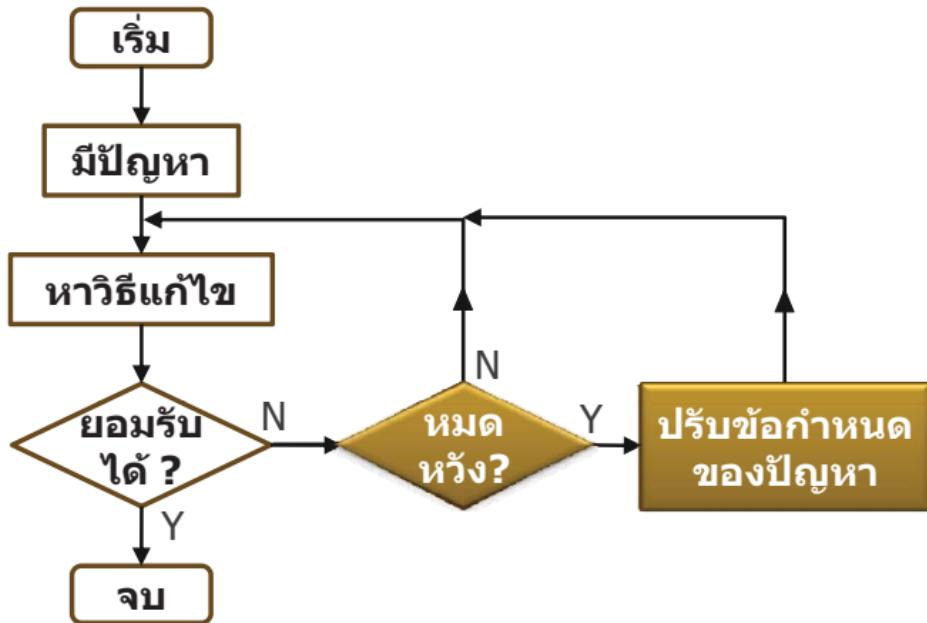
```

hasMajority( d[1..n] )
    for ( i = 1; i <= n; i++ )
        c = 0;
        for ( j = 1; j <= n; j++ )
            if ( d[i] == d[j] ) c++
            if ( c > n/2) return TRUE
    return FALSE

```

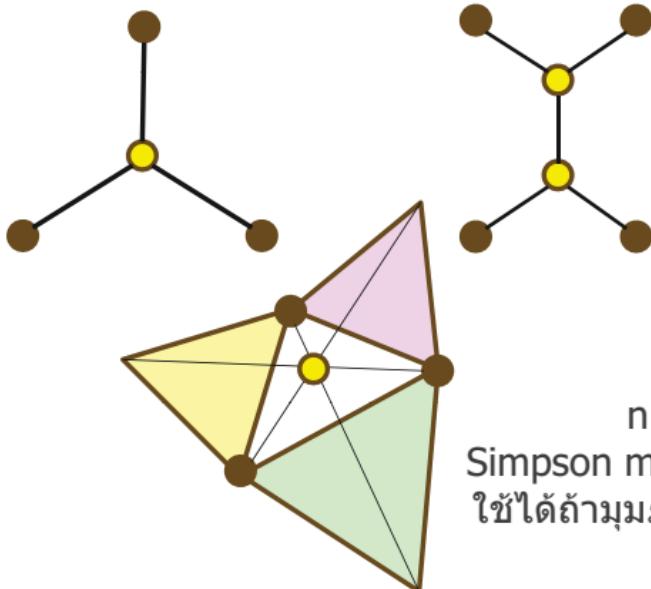
$$\sum_{i=1}^n \left( \sum_{j=1}^n 1 \right) = \sum_{i=1}^n n = n^2$$

# ขั้นตอนการออกแบบอัลกอริทึม



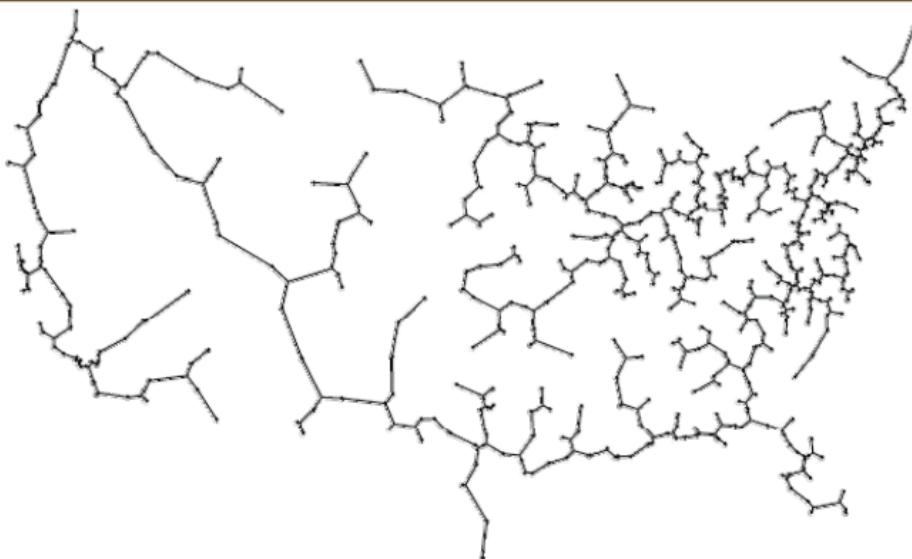
# Steiner Tree

จงหาดต้นไม้ต่อจุดทั้งหลายให้มี  
ความยาวรวมของเส้นเชื่อมที่สั้นสุด



$n = 3$   
Simpson method (1750)  
ใช้ได้ถ้ามุมภายใน  $\leq 120^\circ$

# ข้อมูลจำนวนมาก ใช้เวลานาน

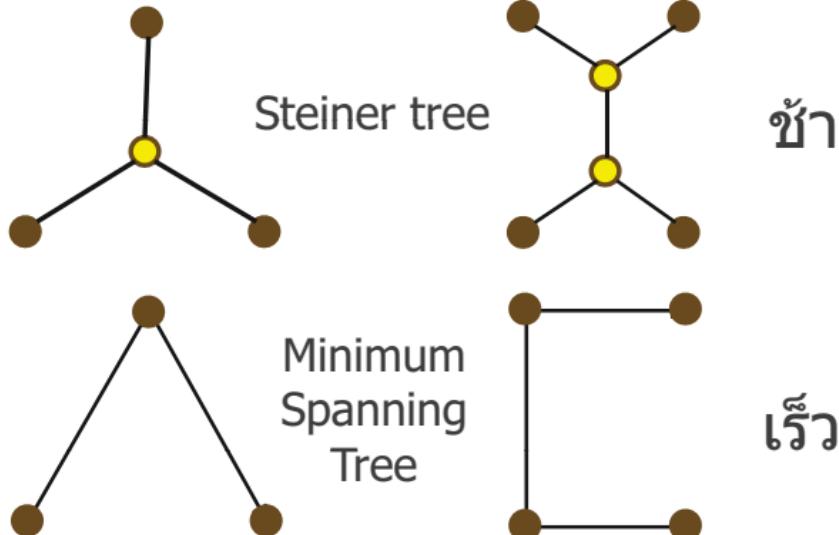


ใช้ GeoSteiner หาคำตอบของปัญหาขนาด 532 ปม  
ใช้เวลา 2.5 ชั่วโมง (ปี ค.ศ. 1998)

<http://www.diku.dk/hjemmesider/ansatte/martinz/geosteiner/>

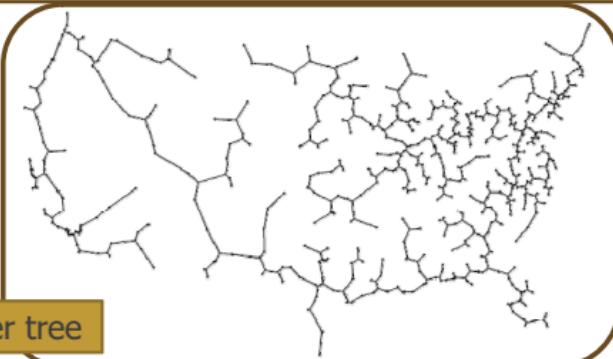
## ปรับข้อกำหนด

Steiner tree → Minimum Spanning tree



$$\text{Length(MST)} \leq 2 \cdot \text{Length(Steiner tree)}$$

# เปรียบเทียบผล



Steiner tree

เป็นชั่วโมง  
สั้นกว่า  $\approx 10\%$



Minimum Spanning Tree

เป็นวินาที

# ถ้าใช้ CPU เร็วขึ้นไม่ได้หรือ ?

- ❖ ได้ในบางกรณี
- ❖ เปลี่ยนเครื่อง เปลี่ยนภาษา เปลี่ยน compiler  
เปลี่ยนโปรแกรมแกรมเมอร์ ก็ได้ แต่ได้ในบางกรณี
- ❖ ต.ย.  $t_1(n) = 10^{-6}n^2$   $n$  คือปริมาณข้อมูล
 
$$t_2(n_2) = t_1(n_1)$$

$$10^{-7} n_2^2 = 10^{-6} n_1^2$$

$$n_2 = (\sqrt{10}) n_1$$
- ❖ ถ้า CPU เร็วขึ้น 10 เท่า
  - ❖ จะได้  $t_2(n) = t_1(n)/10$
  - ❖ ด้วยเวลาเท่าเดิม CPU ใหม่  
จะแก้ปัญหาที่มีปริมาณข้อมูลเพิ่มขึ้นอีก  $\sqrt{10} \approx 3.16$  เท่า

$n$	1000	2000	3000	4000	5000	6000	7000
$t_1(n)$	10	40	90	160	250	360	490
$t_2(n)$	1	4	9	16	25	36	49

# ถ้าซ่า ก็ใช้ CPU เร็วขึ้นไม่ได้หรือ ?

❖ ต.ย.  $t_1(n) = 10^{-10} 2^n$   $n$  คือปริมาณข้อมูล

❖ ถ้า CPU เร็วขึ้น 32 เท่า

❖ ด้วยเวลาเท่าเดิม CPU ใหม่จะแก้ปัญหาที่มีปริมาณข้อมูลเพิ่มขึ้นอีกประมาณ 5 ตัว !!!

$$t_2(n_2) = t_1(n_1) \quad \log_2 2^{n_2} = \log_2(32 \cdot 2^{n_1})$$

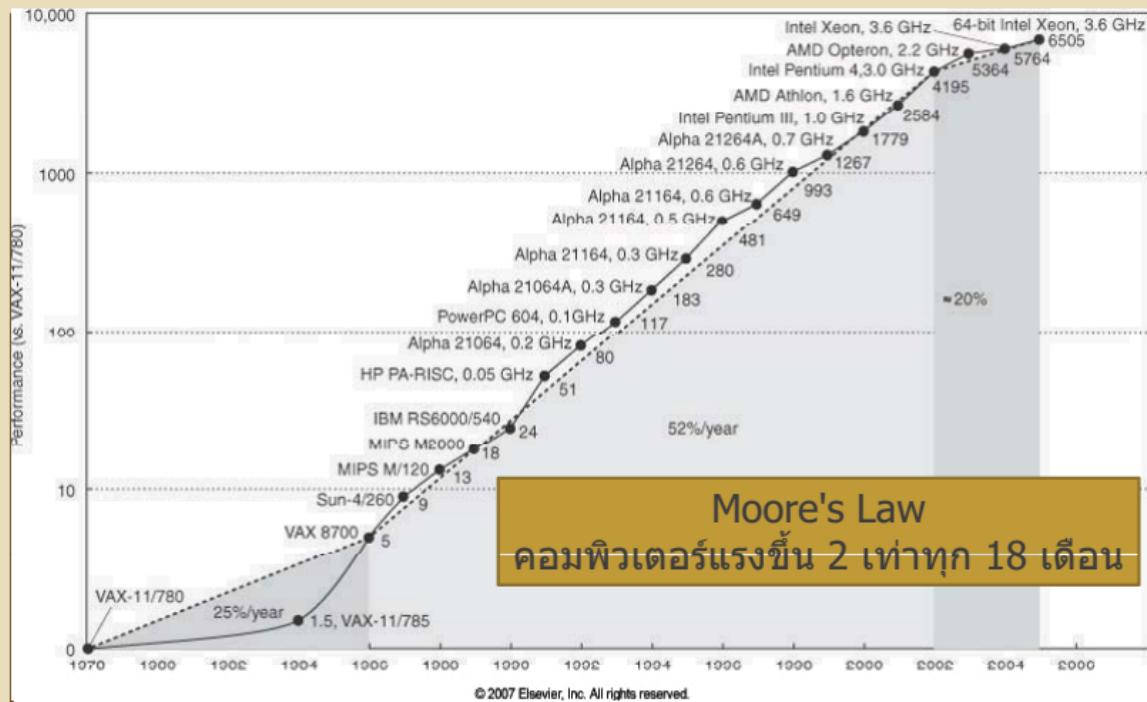
$$\frac{10^{-10} 2^{n_2}}{32} = 10^{-10} 2^{n_1} \quad n_2 = \log_2 32 + \log_2 2^{n_1}$$
$$n_2 = 5 + n_1$$

$$2^{n_2} = 32 \cdot 2^{n_1}$$

$n$	40	45	50	55	100	200	300
$t_1(n)$	$1.1 \times 10^{03}$	$3.5 \times 10^{04}$	$1.1 \times 10^{06}$	$3.6 \times 10^{07}$	$1.3 \times 10^{21}$	$1.6 \times 10^{51}$	$2.0 \times 10^{81}$
$t_2(n)$	$3.4 \times 10^{01}$	$1.1 \times 10^{03}$	$3.5 \times 10^{04}$	$1.1 \times 10^{06}$	$4.0 \times 10^{19}$	$5.0 \times 10^{49}$	$6.4 \times 10^{79}$

$4.0 \times 10^{19}$  มิลลิวินาที  $\approx$  หนึ่งพันล้านปี !!!

# ประสิทธิภาพของหน่วยประมวลผล



# ถ้าเช่า

- ❖ ถ้า  $t(n) = 10^{-10} 2^n$ 
  - ❖ Moore's Law : อีก 7.5 ปี เร็วขึ้น 32 เท่า
  - ❖ แก้ปัญหาได้ขนาดเพิ่มอีกแค่ 5 !!!
- ❖ โดยทั่วไป ถ้าเวลาเป็นฟังก์ชันแบบ exponential ของปริมาณข้อมูล
  - ❖ ต้องเปลี่ยนอัลกอริทึม
  - ❖ ต้องเปลี่ยนข้อกำหนดของปัญหา

# Al-Khwarizmi → Algorithm

- ❖ Abū Abdallāh Muḥammad ibn Mūsā al-Khwārizmī
- ❖ นักคณิตศาสตร์ (ค.ศ. 780 – 850) เขียนหนังสือ
  - ❖ "Algoritmi de numero Indorum" (ภาษาลาติน) แปลว่า "Al-Khwarizmi on the Hindu Art of Reckoning" (อังกฤษ)
  - ❖ นำระบบจำนวนที่มีเลขศูนย์มาใช้ในตะวันออกกลางและยุโรป
  - ❖ เสนอขั้นตอนอย่างมีระบบในการแก้สมการเชิงเส้นและสมการกำลังสอง อันเป็นที่มาของพีชคณิต (algebra)
  - ❖ ...



# สรุป

เข้าใจปัญหา

รู้วิธีออกแบบอัลกอริทึม

รู้วิธีวิเคราะห์

# ทบทวน อัลกอริทึมที่เคยเรียนมา

สมชาย ประสิทธิ์จูตระกูต

# รูปแบบการบรรยายอัลกอริทึม

## ❖ ข้อบรรยายด้วยรหัสเทียม (pseudo-code)

```
gcd(a, b) {  
    while (b > 0) {  
        t = a mod b  
        a = b  
        b = t  
    }  
    return a  
}
```

```
gcd(a, b) {  
    while (b > 0)  
        t = a mod b  
        a = b  
        b = t  
  
    return a  
}
```

# รูปแบบการบรรยายอัลกอริทึม

## ❖ ข้อบรรยายด้วยรหัสเทียม (pseudo-code)

```
seqSearch( d[1..n], x ) {
    for (k = 1; k <= n; k++) {
        if (d[k] == x) return k
    }
    return -1
}
```

```
prim( g[1..v][1..v], s ) {
    t = new array[1..n]
    ...
}
```

## ผลรวม

- ❖ **input** : รายการของจำนวนจริง  $n$  จำนวน  $D = \langle d_1, d_2, d_3, \dots, d_n \rangle$
- ❖ **output** : ผลรวม  $d_1 + d_2 + d_3 + \dots + d_n$

```
sum( d[1..n] ) {  
    s = 0  
    for (k = 1; k <= n; k++) {  
        s += d[k]  
    }  
    return s  
}
```

$$s = \sum_{k=1}^n d_k$$

# ค่ามากสุด

- ❖ **input** : รายการของจำนวนจริง  $n$  จำนวน  
 $D = \langle d_1, d_2, d_3, \dots, d_n \rangle$
- ❖ **output** : ค่ามากสุดของ  $d_1, d_2, d_3, \dots, d_n$

```
max( d[1..n] ) {
    m = d[1]
    for (k = 2; k <= n; k++) {
        if (d[k] > m) m = d[k]
    }
    return m
}
```

$$m = \max_{1 \leq k \leq n} \{ d_k \}$$

# ตำแหน่งของค่ามากสุด

- ❖ **input** : รายการของจำนวนจริง  $n$  จำนวน $D = \langle d_1, d_2, d_3, \dots, d_n \rangle$
- ❖ **output** : ตำแหน่ง **max**  $d_{\max} = \max_{1 \leq k \leq n} \{ d_k \}$

```
max( d[1..n] ) {  
    maxI = 1  
    for (k = 2; k <= n; k++) {  
        if (d[k] > d[maxI]) maxI = k  
    }  
    return maxI  
}
```

# การค้นแบบลำดับ

- ❖ **input** :  $x$  และ  $D = \langle d_1, d_2, d_3, \dots, d_n \rangle$   
 $x$  และ  $d_i$  เป็นจำนวนจริง
- ❖ **output** : ค่า  $k$  ที่  $d_k = x$  ถ้าหาไม่พบ คืน  $-1$

```
seqSearch( d[1..n], x ) {  
    for (k = 1; k <= n; k++) {  
        if (d[k] == x) return k  
    }  
    return -1  
}
```

## การค้นแบบลำดับ

- ❖ **input** :  $x$  และ  $D = \langle d_1, d_2, d_3, \dots, d_n \rangle$   
 $d_1 \leq d_2 \leq d_3 \leq \dots \leq d_n$   
 $x$  และ  $d_i$  เป็นจำนวนจริง
- ❖ **output** : ค่า  $k$  ที่  $d_k = x$  ถ้าหาไม่พบ คืน  $-1$

```
seqSearch( d[1..n], x ) {  
    for (k = 1; k <= n; k++) {  
        if (d[k] == x) return k  
        if (d[k] > x) return -1  
    }  
    return -1  
}
```

D = <2, 3, 6, 8, 9, 10, 29>  
x = 7

# การค้นแบบทวิภาค (binary search)

- ❖ **input** :  $x$  และ  $D = \langle d_1, d_2, d_3, \dots, d_n \rangle$   
 $d_1 \leq d_2 \leq d_3 \leq \dots \leq d_n$   
 $x$  และ  $d_i$  เป็นจำนวนจริง
- ❖ **output** : ค่า  $k$  ที่  $d_k = x$  ถ้าหากไม่พบ คืน -1

	1	2	3	4	5	6	7	8	9	10	11	12
$d$	2	3	5	9	11	20	25	39	44	49	52	79

$x = 25$

	1	2	3	4	5	6	7	8	9	10	11	12
$d$	2	3	5	9	11	20	25	39	44	49	52	79

$x = 27$

# การค้นแบบทวิภาค

```
binarySearch( d[1..n] , x ) {  
    left = 1, right = n  
    while ( left <= right ) {  
        mid = ⌊(left + right) / 2⌋  
        if (x == d[mid]) return mid  
        if (x < d[mid])  
            right = mid - 1  
        else  
            left = mid + 1  
    }  
    return -1;  
}
```

# Selection Sort

```
selectionSort( d[1..n] ) {  
    for (k = n; k >= 2; k--) {  
        m = maxI(d, k)  
        d[k] ↔ d[m]  
    }  
    maxI( d[1..n], k ) {  
        maxI = 1  
        for (i = 2; i <= k; i++)  
            if (d[i] > d[maxI]) maxI = i  
        return maxI  
    }  
}
```

# คู่ใดบวกกันได้ x

- ❖ **input** : x และ D =  $\langle d_1, d_2, d_3, \dots, d_n \rangle$   
x และ  $d_i$  เป็นจำนวนจริง
- ❖ **output** : i และ j โดยที่  $i \neq j$  และ  $(d_i + d_j) = x$   
ถ้าไม่พบคืน -1, -1

x = 7    D = < 9, 2, 4, 3, 7, 5, 0 >

```
sum2( d[1..n], x ) {  
    for (i = 1; i <= n; i++)  
        for (j=i+1; j <= n; j++)  
            if ( [ ] d[i]+d[j] == x) return i,j  
    return -1, -1  
}
```

# คู่ๆ ได้บวกกันได้ x

- ❖ **input** :  $x$  และ  $D = \langle d_1, d_2, d_3, \dots, d_n \rangle$   
 $x$  และ  $d_i$  เป็นจำนวนจริง
- ❖ **output** :  $i$  และ  $j$  โดยที่  $i \neq j$  และ  $(d_i + d_j) = x$   
ถ้าไม่พบคืน  $-1, -1$

$x = 7$     $D = \langle 9, 2, 4, 3, 7, 5, 0 \rangle$

หา  $7 - 9 = -2$

หา  $7 - 2 = 5$

$x = 7$     $D = \langle 0, 2, 3, 4, 5, 7, 9 \rangle$

หา  $7 - 0 = 7$

ถ้า  $D$  เรียงลำดับแล้ว ก็ค้นด้วย binary search

# คู่ใดบวกกันได้ x

- ❖ **input** : x และ  $D = \langle d_1, d_2, d_3, \dots, d_n \rangle$   
 $d_1 \leq d_2 \leq d_3 \leq \dots \leq d_n$   
x และ  $d_i$  เป็นจำนวนจริง
- ❖ **output** : i และ j โดยที่  $i \neq j$  และ  $(d_i + d_j) = x$   
ถ้าไม่พบคืน -1, -1

```
sum2( d[1..n], x ) {
    for (i = 1; i <= n; i++) {
        delta = x - d[i]
        j = binarySearch( d, delta )
        if ( j != -1 ) return i, j
    }
    return -1,-1
}
```

# อัลกอริทึมที่เขียนจากสูตร

$$x_k = \frac{1}{2} \left( x_{k-1} + \frac{a}{x_{k-1}} \right), \quad x_0 = 1$$

$$\sqrt{a} = \lim_{k \rightarrow \infty} x_k$$

<b>a</b>	= 100
<b>x<sub>0</sub></b>	= 1.0
<b>x<sub>1</sub></b>	= 50.5
<b>x<sub>2</sub></b>	= 26.24
<b>x<sub>3</sub></b>	= 15.03
<b>x<sub>4</sub></b>	= 10.84
<b>x<sub>5</sub></b>	= 10.03
<b>x<sub>6</sub></b>	= 10.00

```

sqrt( a ) {
    x = 1
    while ( x2 ≠ a )
        x = (x + a/x)/2
    return x
}

```

$|x^2 - a| > 10^{-10}$

# อัลกอริทึมที่เขียนจากสูตร

```
gcd(a, b) {           gcd(a, b) = gcd(b, a mod b)
    while (b > 0) {
        t = a mod b   gcd(a, 0) = a
        a = b          gcd(45, 24) = gcd(24, 21)
        b = t          = gcd(21, 3)
    }                = gcd(3, 0)
    return a          = 3
}
```

# อัลกอริทึมทำซ้ำ แบบวนและแบบเรียกซ้ำ

สมชาย ประสิทธิ์ยุตระกูต

# หัวข้อ

**sum**

**max**

**sequential search**

**binary search**

**gcd (Euclid)**

**logarithm**

**binary counter**

**sqrt (Babylonian)**

**sum of subset**

# Iterative & Recursive

```
sum( d[1..n] ) {
    s = 0
    for (k = 1; k <= n; k++) {
        s += d[k]
    }
    return s
}
```

$$S = \sum_{k=1}^n d_k$$

```
sum( d[1..n] ) {
    return s(d, n)
}
s( d[1..n], k ) {
    if ( k == 0 ) return 0
    return s(d, k-1) + d[k]
}
```

$$\begin{aligned} s_k &= s_{k-1} + d_k, \text{ if } k > 0 \\ s_0 &= 0 \end{aligned}$$

# max

```
max( d[1..n] ) {  
    m = d[1]  
    for (k = 2; k <= n; k++) {  
        if (d[k] > m) m = d[k]  
    }  
    return m  
}
```

$$m = \max_{1 \leq k \leq n} \{ d_k \}$$

```
max( d[1..n] ) {  
    return m(d, n)  
}  
  
m( d[1..n], k ) {  
    if ( k == 1 ) return d[1]  
    return max( m(d, k-1), d[k] )  
}
```

$$\begin{aligned} m_k &= \max(m_{k-1}, d_k), \quad \text{if } k > 1 \\ m_1 &= d_1 \end{aligned}$$

# Sequential Search

```
seqSearch( d[1..n], x ) {  
    for (k = 1; k <= n; k++) {  
        if (d[k] == x) return k  
    }  
    return -1  
}
```

```
seqSearch( d[1..n], x ) {  
    return search( d, x, n )  
}  
search( d[1..n], x, k ) {  
    if ( k == 0 ) return -1  
    if ( d[k] == x ) return k  
    return search( d, x, k-1 )  
}
```

# Binary Search

```
binarySearch( d[1..n], x ) {
    return bsearch( d, x, 1, n )
}
bsearch( d[1..n], x, left, right ) {
    if (left > right) return -1
    mid = ⌊(left + right) / 2⌋
    if (x = d[mid]) return mid
    if (x < d[mid])
        return bsearch( d, x, left, mid - 1)
    else
        return bsearch( d, x, mid + 1, right )
}
```

# Greatest Common Divisor

```

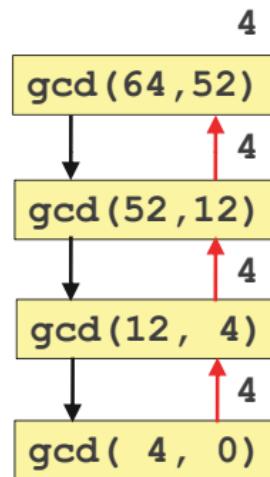
gcd(a, b) {
    while (b > 0) {
        t = a mod b
        a = b
        b = t
    }
    return a
}

```

$\text{gcd}(a,b) = \text{gcd}(b,a \bmod b)$

$\text{gcd}(a,0) = a$

```
gcd(a, b) {  
    if (b == 0) return a  
    return gcd(b, a mod b)  
}
```



# $\lfloor \log_2 n \rfloor$

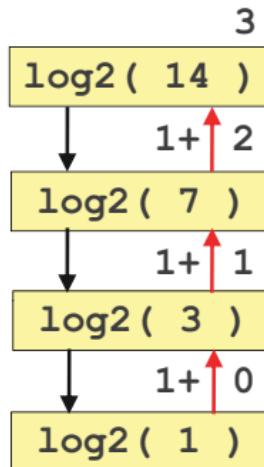
```
log2( n ) {  
    c = 0  
    while (n > 1) {  
        n = ⌊n / 2⌋  
        c++  
    }  
    return c  
}
```

n	$\lfloor \log_2 n \rfloor$
1	0
2	1
3	1
4	2
5	2
6	2
7	2
8	3
9	3

```
log2( n ) {  
    if ( n <= 1 ) return 0  
    return 1 + log2( ⌊n/2⌋ )  
}
```

# $\lfloor \log_2 n \rfloor$

```
log2( n ) {  
    if ( n <= 1 ) return 0  
    return 1 + log2(n/2)  
}
```



# Binary Counter

```
bcounter( n ) {  
    d = array of size n  
    for (i = 1; i <= n; i++) d[i] = 0  
    for (i = 1; i <= 2n; i++) {  
        print( d )  
        increment( d )  
    }  
}  
increment( d[1..n] )  
for (i=n; i>=1 AND d[i]=1; i--)  
    d[i] = 0  
    if (i >= 1) d[i] = 1  
}
```

<0,0,1,1,1>  
↓ ↓ ↓ ↓  
<0,1,0,0,0>

0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

# Binary Counter : Recursive

```

bcounter( n ) {
    d = array of size n
    for (i = 1; i <= n; i++) d[i] = 0
    count( d, 0 )
}
count( d[1..n], k ) {
    if ( k = n ) print( d )
    else {
        d[k+1] = 0; count( d, k+1 )
        d[k+1] = 1; count( d, k+1 )
    }
}
    
```

k = 2      

0	1	?	?	?
---	---	---	---	---

k = 3      

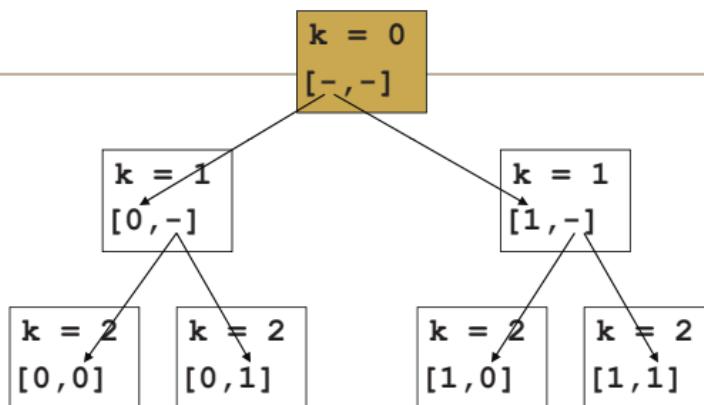
0	1	0	?	?
---	---	---	---	---

k = 3      

0	1	1	?	?
---	---	---	---	---

# Binary Counter : Recursive

```
count( d[1..n] , k ) {  
    if ( k = n ) print( d )  
    else {  
        d[k+1] = 0; count( d, k+1 )  
        d[k+1] = 1; count( d, k+1 )  
    }  
}
```



# sqrt

```

sqrt( a ) {
    x = 1
    while ( x2 ≠ a )
        x = (x + a/x)/2
    return x
}

```

$$|x^2 - a| \geq 10^{-10}$$

$$x_k = \frac{1}{2} \left( x_{k-1} + \frac{a}{x_{k-1}} \right), \quad x_0 = 1$$

```

sqrt( a ) {
    return sqrt(a, 1);
}
sqrt( a, x ) {
    if (|x2 - a| < 10-10) return x;
    return sqrt( a, (x + a/x)/2 );
}

```

## Sum of Subset

- ❖ **input** :  $x$  และ  $D = \{d_1, d_2, d_3, \dots, d_n\}$   
 $x$  และ  $d_i$  เป็นจำนวนจริง
- ❖ **output** : true ถ้า  $D$  มีเซตย่อยที่ผลรวมเท่ากับ  $x$   
ถ้าไม่มี คืน false

$D = \{1, 4, 3, 9, 7\}, \quad x = 11 \quad \text{true}$

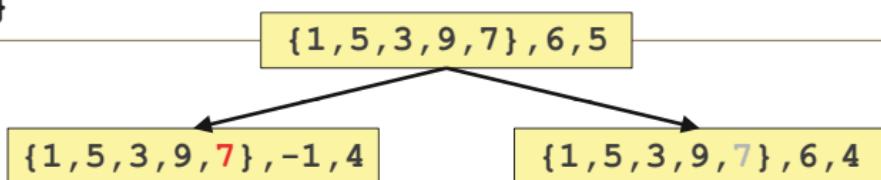
$D = \{1, 4, 3, 9, 7\}, \quad x = 18 \quad \text{false}$

# Sum of Subset

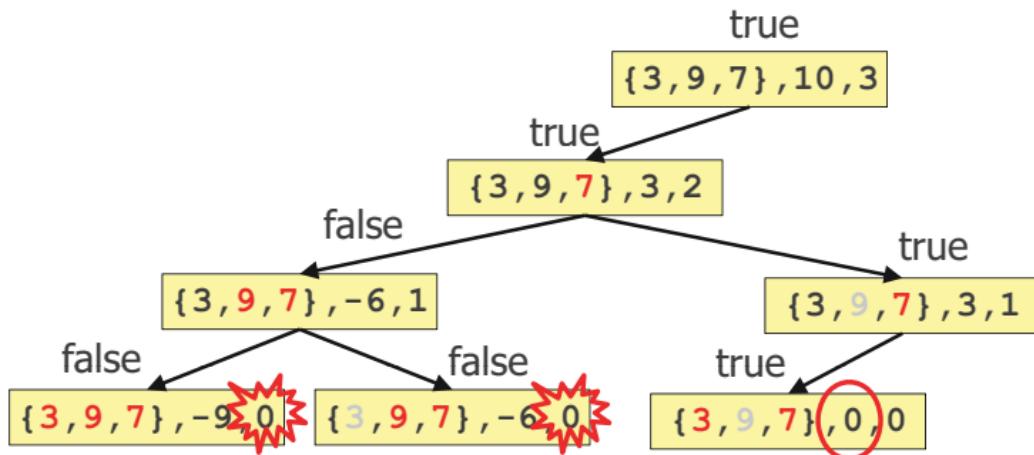
```

subsetSum( d[1..n], x ) {
    return subsetSum( d, x, n )
}
subsetSum( d[1..n], x, k ) {
    if (x == 0) return true;
    if (k < 1) return false;
    return subsetSum(d, x-d[k], k-1) OR
           subsetSum(d, x, k-1);
}

```



$$D = \{3, 9, 7\}, x = 10$$



# ทบทวน โครงสร้างข้อมูล

สมชาย ประสิทธิจูตระกูต

# หัวข้อ

- ❖ **Abstract Data Types**

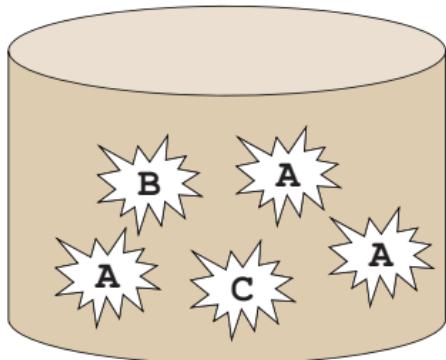
- ❖ Collection, Set, List, Stack, Queue, Priority Queue

- ❖ **Implementations**

- ❖ array-based stack and queue, linked list, binary search tree, AVL tree, hash table (separate chaining and open addressing), binary heap

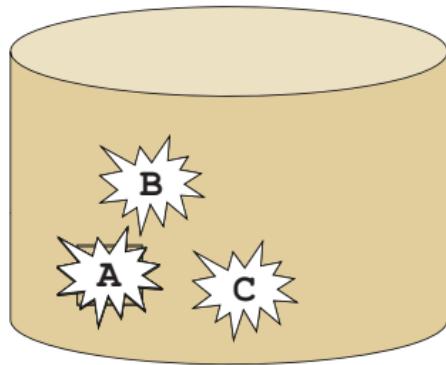
# Collection

❖ เก็บข้อมูลไม่มีอันดับ ข้าໄດ້



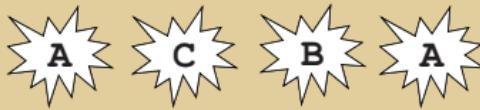
# Set

❖ เก็บข้อมูลไม่มีอันดับ 'ไม่ให้ช้า'



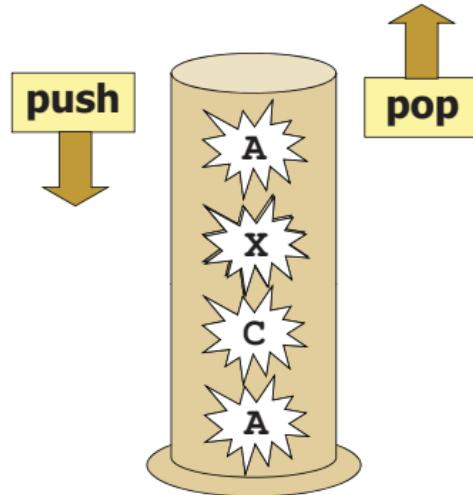
# List

❖ เก็บข้อมูลเรียงแบบมีอันดับ ข้าไป



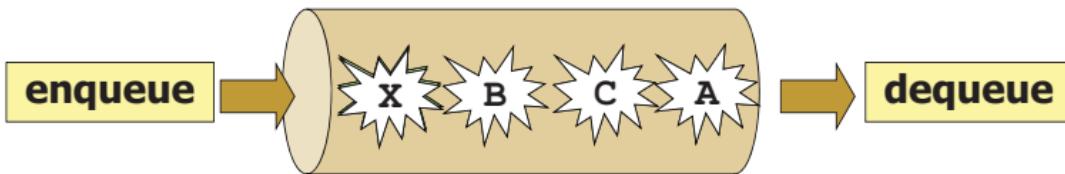
# Stack

- ❖ ข้อมูล เข้าหลัง ออกก่อน  
**(LIFO - Last-In First-Out)**



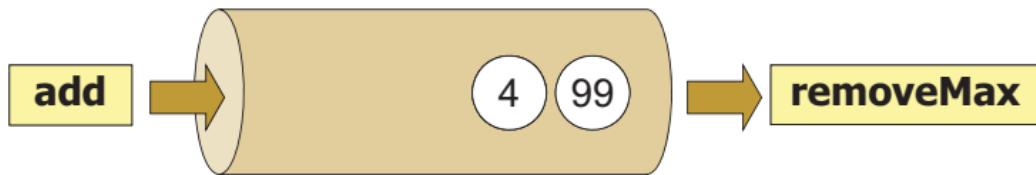
# Queue

- ❖ ຂໍ້ມູນ ເຂົາກ່ອນ ອອກກ່ອນ  
**(FIFO - First-In First-Out)**



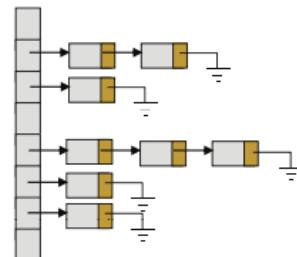
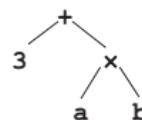
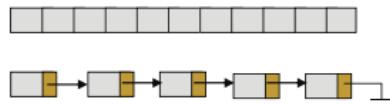
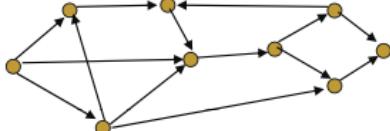
# Priority Queue

- ❖ ออกรดตามความสำคัญ (แกะค้อยลดคิวได้)



# วิธีสร้างที่เก็บข้อมูล

- ❖ สร้างด้วยอาร์เรย์
- ❖ สร้างด้วยการโยง
- ❖ สร้างด้วยต้นไม้
- ❖ สร้างด้วยตาราง
- ❖ อื่น ๆ



# การสร้าง list ด้วยอาร์เรย์



X

add(1, X)

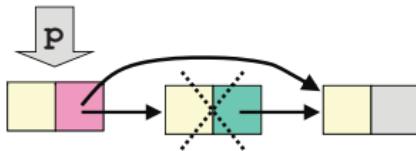
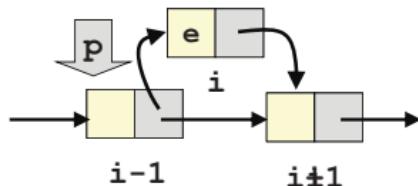
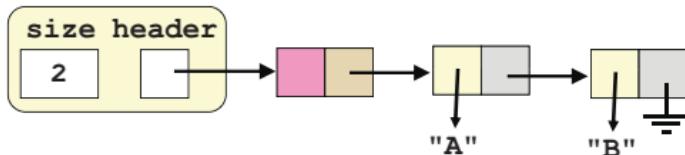
ข้อมูลเดิมอาร์เรย์ก็ขยาย  
หรือไม่ก็จะงเพื่อไว



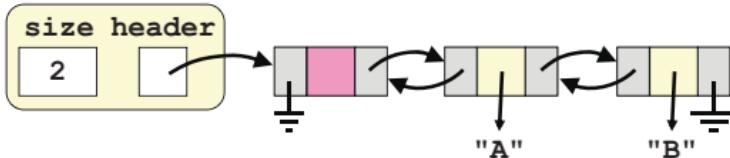
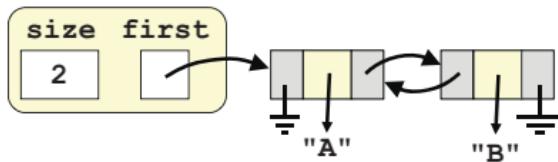
remove(0)

เวลา  $\propto$  ปริมาณข้อมูล  
 $t \propto n$

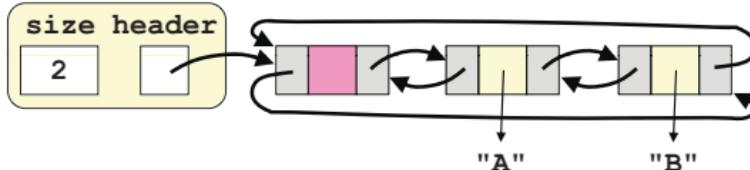
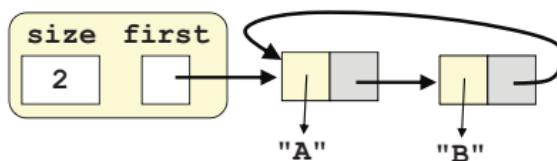
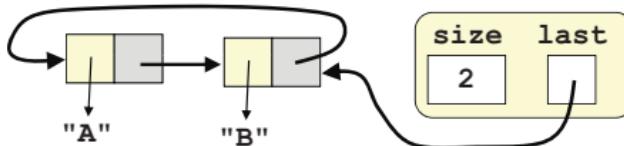
# การสร้าง list แบบ linked list



# Linked list แบบโยงไป-กลับ



# Linked list แบบไซจวน (circular)



# สร้าง Stack ด้วยอาเรย์

Stack s	size	elementData
s = new Stack(4)	0	
s.push("A")	1	A
s.push("B")	2	A B
s.push("C")	3	A B C
s.pop()	2	A B

ข้อมูลเต็มอาเรย์ก็ขยาย  
หรือไม่ก็จะคงเพื่อไว้

ใช้เวลาคงตัว

# สร้าง Queue ด้วยอาร์เรย์

Queue q                  front size elementData

q = new Queue(4)

	0	1	2	3
front	0	1	2	3

q.enqueue("A")

0	1	A		

q.enqueue("B")

0	2	A	B	

q.dequeue()

1	1		B	

q.enqueue("C")

1	2		B	C

q.dequeue()

2	1			C

q.enqueue("D")

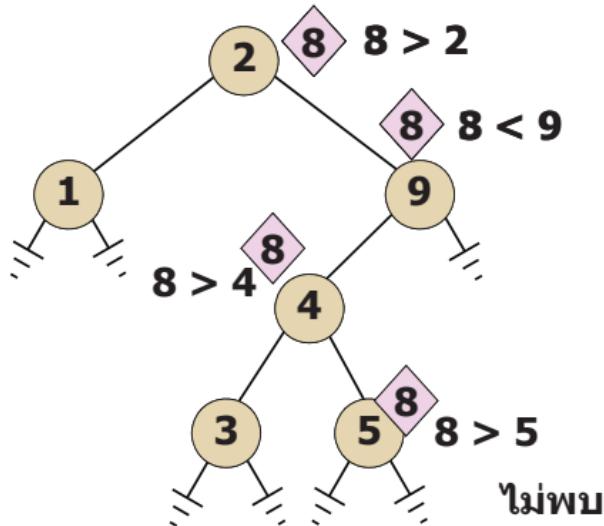
2	2		C	D

q.enqueue("X")

2	3	X	C	D

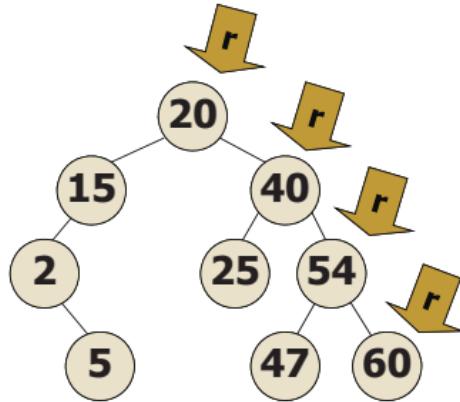
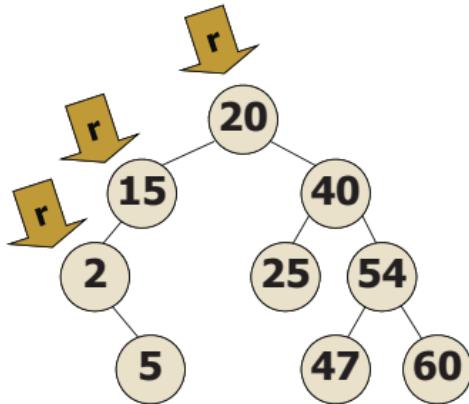
ใช้เวลาคงตัว

# Binary Search Tree



$$t \propto h$$

# BST : min, max



$$t \propto h$$

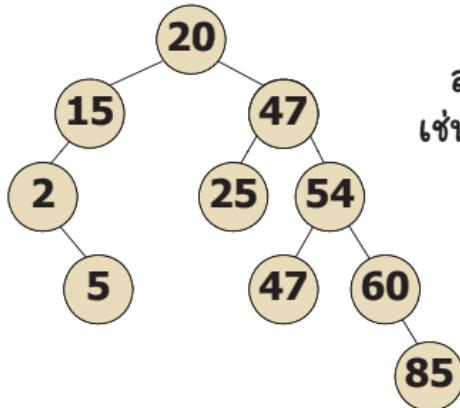
# สร้าง BST ด้วยการค่อย ๆ เพิ่ม

1, 2, 6, 3, 5

2, 1, 5, 3, 6

ความสูงของต้นไม้มีขึ้นกับลำดับ  
ของข้อมูลที่เพิ่มใส่ต้นไม้

## BST : remove



ลบปมที่มี 2 ลูก  
 เช่น ต้องการลบ 40

ลบใบ  
 เช่น ต้องการลบ 5

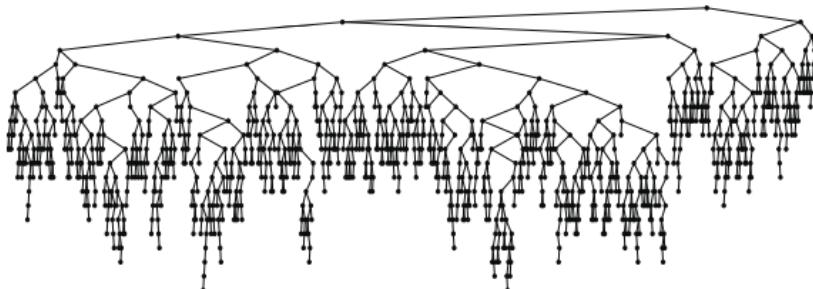
ลบปมที่มีลูกเดียว  
 เช่น ต้องการลบ 60

$$t \propto h$$

# BST ที่สร้างจากข้อมูลสุ่ม

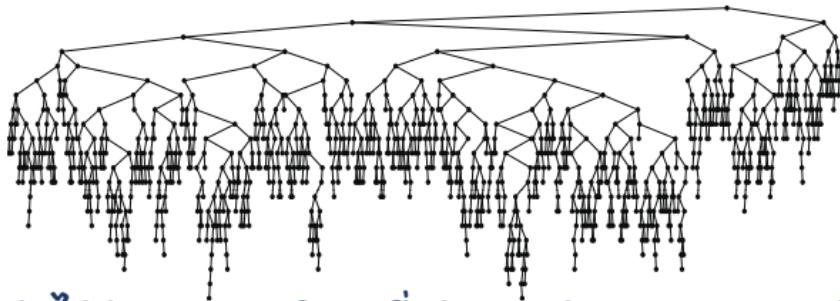
- ❖ ต้นไม้ที่เก็บข้อมูล  $n$  ตัว
- ❖ ช่วงของความสูง :  $\lfloor \log_2 n \rfloor \leq h \leq n - 1$
- ❖ ถ้าสร้างจากข้อมูลสุ่ม สามารถวิเคราะห์ได้ว่า
  - ❖ ความลึกเฉลี่ยของปมภายใน  $\approx 1.39 \log_2 n$
  - ❖ ความลึกเฉลี่ยของ null  $\approx 2 + 1.39 \log_2 n$
  - ❖ ความสูง (ความลึกของใบล่างสุด)  $\approx 2.99 \log_2 n$

$$t \propto h$$

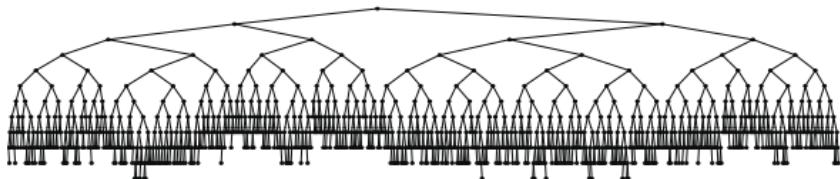


Devroye, L. 1986. A note on the height of binary search trees. *J. ACM* 33, 489–498.

## BST vs. AVL



ต้นไม้มีคันหาแบบทวิภาคที่สร้างจากข้อมูลสุ่ม 1000 ตัว



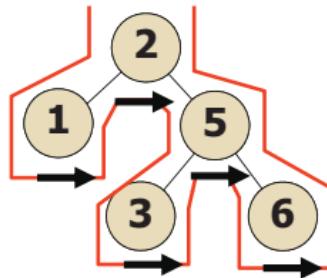
ต้นไม้อาวแอลที่สร้างจากข้อมูลสุ่ม 1000 ตัว

$$\lfloor \log_2 n \rfloor \leq h < 1.44 \log_2 n$$

# การเรียงลำดับข้อมูลแบบต้นไม้

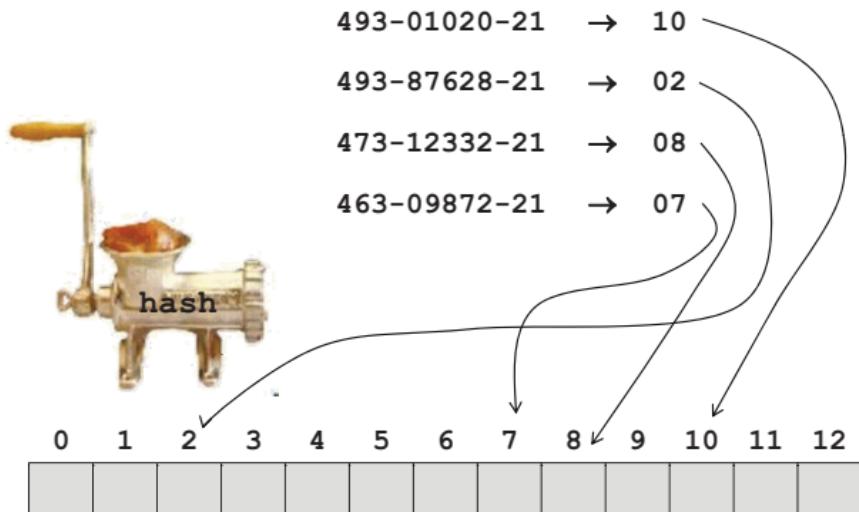
- ❖ นำข้อมูลทั้งหมด มาสร้างต้นไม้คันหาแบบหิวภาค
- ❖ แบ่งผ่านต้นไม้นี้แบบตามลำดับ

2, 1, 5, 3, 6

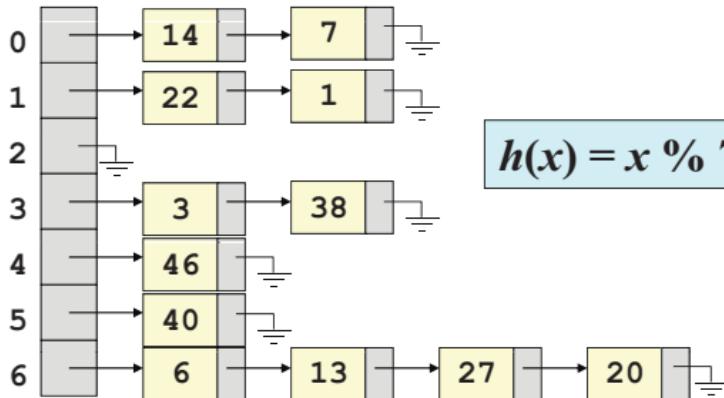


1 2 3 5 6

# Hash Table



# Hash Table : Separate Chaining



$$t \propto \frac{n}{\text{ขนาดตาราง}}$$

# Hash Table : Linear Probing

0	1	2	3	4	5	6	7	8	9	10	11	12

ใช้  $h(x) = x \% 13$  และเพิ่มข้อมูลที่มีคีย์ตามลำดับดังนี้

17    32    26    7    4    43    12    11    24

# Hash Table : Quadratic Probing

0	1	2	3	4	5	6	7	8	9	10	11	12

ใช้  $h(x) = x \% 13$  และเพิ่มข้อมูลที่มีคีย์ตามลำดับดังนี้

4      5      8      0      7      1      17

# เปรียบเทียบเวลาการทำงาน

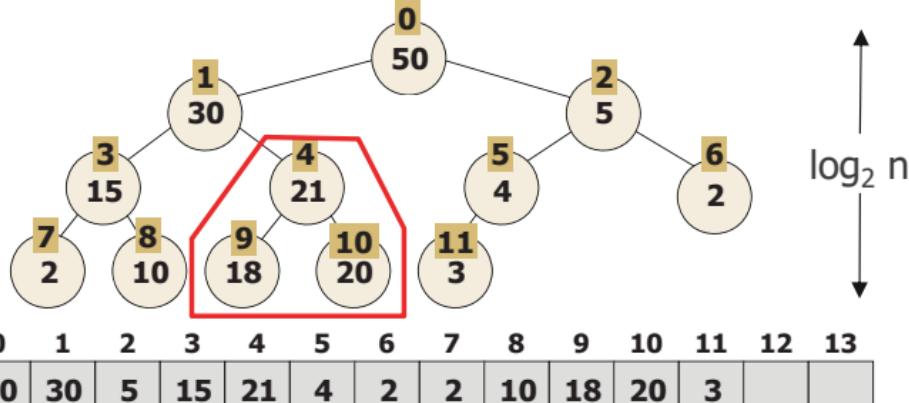
$1117 = 1 \times 3 \times 3 \times 3 / 2 \times 3 \times 3 \times 3 \times 3 / 2 / 2 \times 3 \times 3 / 2 / 2 / 2 / 2 \times 3 \times 3 / 2 / 2 / 2 \times 3 / 2$

```
public static void main(String[] args) {
    Set set = new ArraySet();
```

สร้าง Set ด้วย	เวลาการทำงาน (ms)
ArraySet	164987
BSTSet	1112
AVLSet	430
LinearProbingHashSet	1903
QuadraticProbingHashSet	390
SeparateChainingHashSet	350

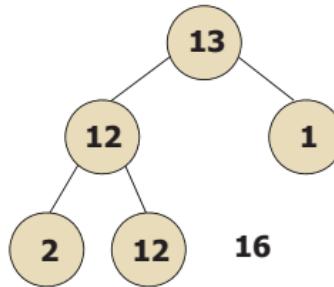
ตอนทำงานเสร็จ set มีข้อมูลจำนวน **73816** ตัว

# Priority Queue : Binary Heap



- รากเก็บที่ index 0
- ลูกซ้ายของ node ที่ index k อยู่ที่ index  $2k + 1$
- ลูกขวาของ node ที่ index k อยู่ที่ index  $2k + 2$
- พ่อของ node ที่ index k อยู่ที่ index  $(k - 1) / 2$

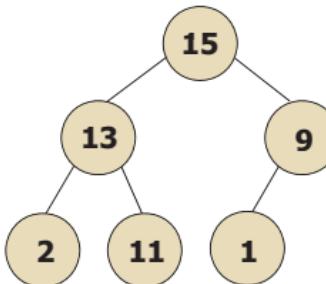
# Binary Heap : add



size	0	1	2	3	4	5	6
	13	12	1	2	12		

$$t \propto \log_2 n$$

# Binary Heap : removeMax



size	6	0	1	2	3	4	5	6
		15	13	9	2	11	1	

elementData

$$t \propto \log_2 n$$

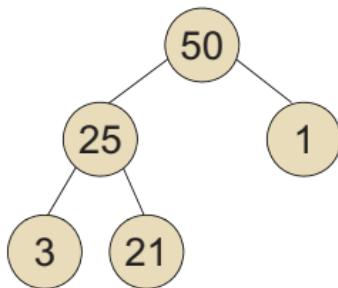
# MaxHeap Min Heap

## ❖ ฮีปมากสุด

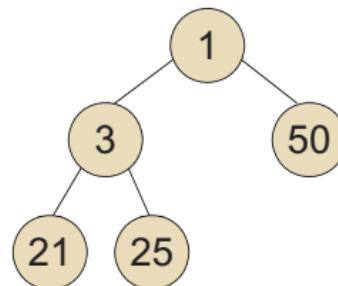
- ❖ ข้อมูลของ parent node มีค่า มากกว่า ของลูก ๆ

## ❖ ฮีปน้อยสุด

- ❖ ข้อมูลของ parent node มีค่า น้อยกว่า ของลูก ๆ

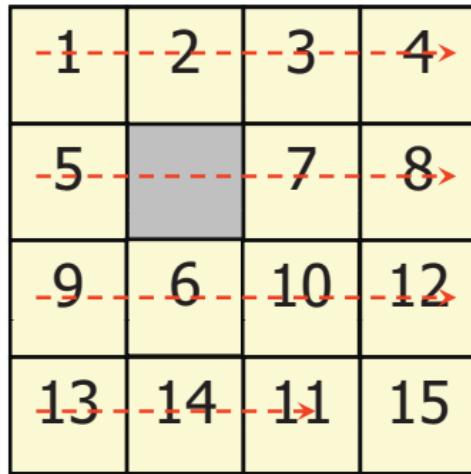


max heap

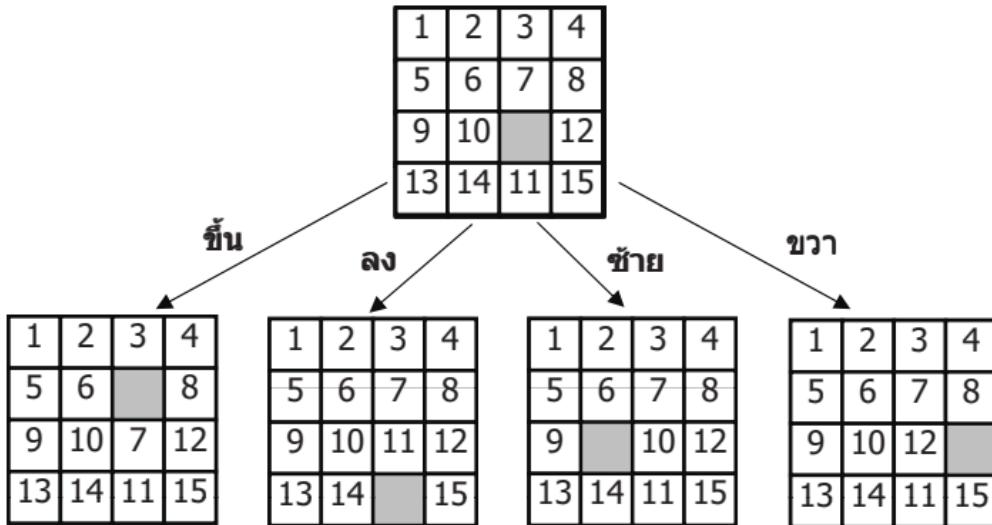


min heap

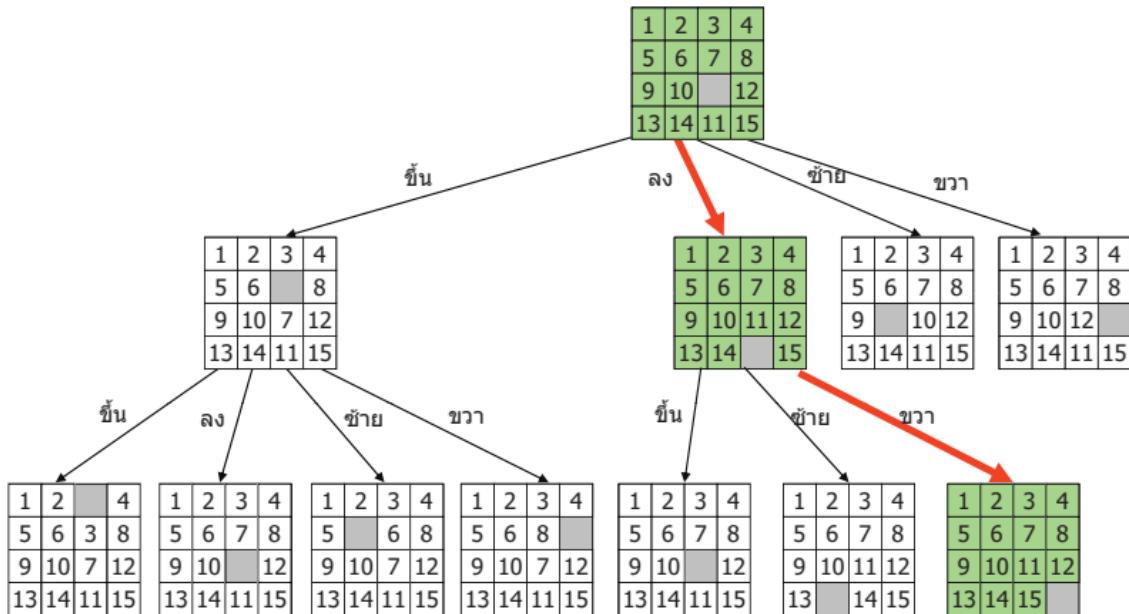
## ตัวอย่าง : 15-puzzle



# เลื่อนช่องว่างเพื่อผลิตตราง



# ພບຄຳຕອບ ພບວິທີເລື່ອນ



# 15-puzzle : ส่วนของโปรแกรม

```

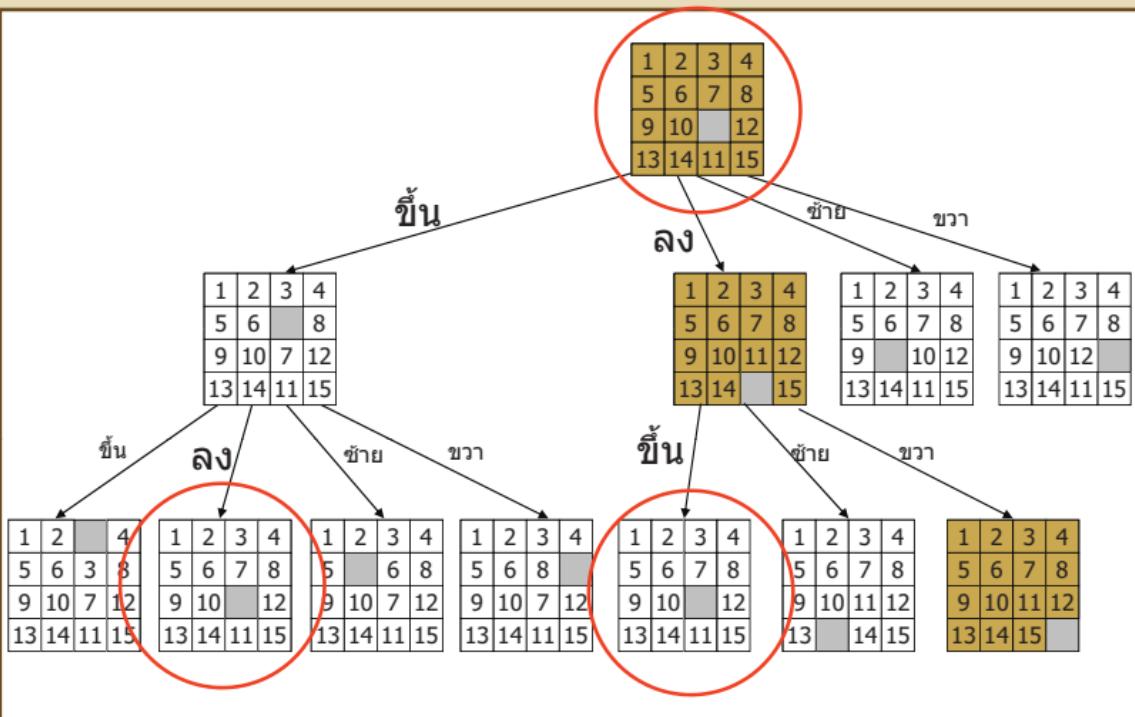
public static PuzzleBoard solve(PuzzleBoard b) {
    Queue queue = new ArrayQueue();
    queue.enqueue(b);
    while ( !queue.isEmpty() ) {
        b = queue.dequeue();
        for (int d = 0; d < 4; d++) {
            PuzzleBoard b2 = b.moveBlank(d);
            if (b2 != null) {
                if ( b2.isAnswer() ) return b2;

                queue.enqueue(b2);
            }
        }
    }
    return null;
}

```

ใช้ queue เก็บตารางที่ผลิตใหม่

# 15-puzzle : ตารางที่ผลิตอาเจ้ากัน



# 15-puzzle : ส่วนของโปรแกรม

```

public static PuzzleBoard solve(PuzzleBoard b) {
    Set set = new HashSet();
    Queue queue = new ArrayDeque();
    queue.enqueue(b);
    set.add(b);
    while (!queue.isEmpty()) {
        b = queue.dequeue();
        for (int d = 0; d < 4; d++) {
            PuzzleBoard b2 = b.moveBlank(d);
            if (b2 != null) {
                if (b2.isAnswer()) return b2;
                if (!set.contains(b2)) {
                    queue.enqueue(b2);
                    set.add(b2);
                }
            }
        }
    }
    return null;
}

```

ใช้ queue เก็บตารางที่ผลิตใหม่

ใช้ set เพื่อตรวจสอบความซ้ำซ้อน

# สร้าง Set ได้หลายแบบ

```
public static PuzzleBoard solve(PuzzleBoard b) {  
    Set set = new HashSet();   
    Queue queue = new ArrayDeque();  
    queue.enqueue(b); set.add(b);  
    while ( !queue.isEmpty() ) {  
        b = queue.dequeue();  
        for (int d = 0; d < 4; d++) {  
            PuzzleBoard b2 = b.moveBlank(d);  
            if (b2 != null) {  
                if ( b2.isAnswer() ) return b2;  
                if ( !set.contains(b2) ) {  
                    queue.enqueue(b2); set.add(b2);  
                }  
            }  
        }  
    }  
    return null;  
}
```

# ผลการทดลอง

ตาราง เริ่มต้น	จำนวนตาราง ที่ผลิต	เวลาการทำงาน (วินาที)			
		ArraySet	BSTSet	AVLSet	HashSet
แบบที่ 1	552	0.03	0.02	0.04	0.05
แบบที่ 2	5242	1.94	0.22	0.18	0.12
แบบที่ 3	132049	1819.6	7.08	5.71	2.56

# สรุป

## ❖ List

- ❖ array :  $\propto n$ , linked) :  $\propto n$ , constant

## ❖ Search Tree

- ❖ BST :  $\propto \log n$  (avg), AVL  $\propto \log n$

## ❖ Stack, Queue

- ❖ push, pop, peek, enqueue, dequeue, front : constant

## ❖ Binary Heap

- ❖ add, removeMax :  $\propto \log n$ , findMax : constant

## ❖ Hash Table

- ❖ add, remove, find :  $\propto (n / T)$

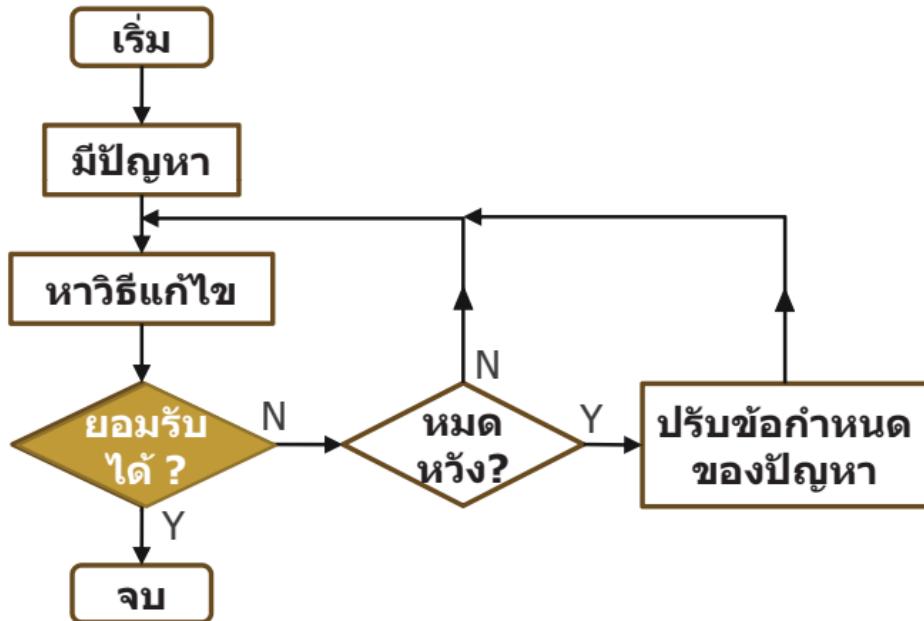
# การวิเคราะห์อัลกอริทึม

สมชาย ประสิทธิ์จูตระกูต

# หัวข้อ

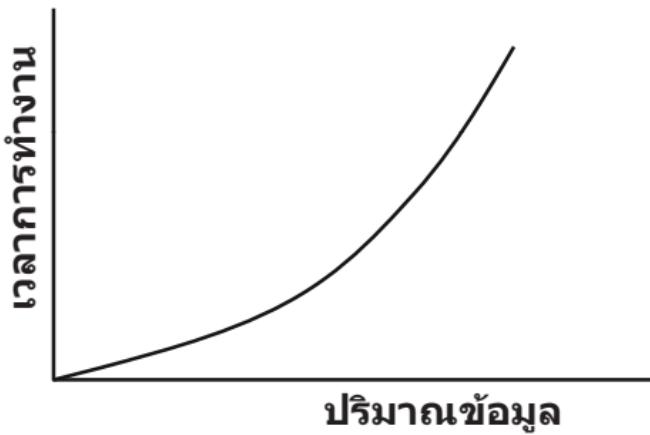
- ❖ การวิเคราะห์อัลกอริทึม
  - ❖ แบบทดลอง และแบบคณิตวิเคราะห์
- ❖ การนับจำนวนการทำงานของคำสั่งพื้นฐาน
- ❖ อัตราการเติบโตของฟังก์ชัน
- ❖ สัญกรณ์เชิงเส้นกำกับ
- ❖ การวิเคราะห์การทำงาน
  - ❖ แบบลำดับ, แบบเลือกทำ, แบบวงวน, แบบเรียกซ้ำ

# ขั้นตอนการออกแบบอัลกอริทึม



# การวิเคราะห์อัลกอริทึม

- ❖ เพื่อศึกษาประสิทธิภาพของอัลกอริทึม
  - ❖ เวลาการทำงาน
  - ❖ ปริมาณหน่วยความจำที่ใช้ในการทำงาน
- ❖ จะเน้นวิเคราะห์เฉพาะเวลาการทำงาน



# การวิเคราะห์อัลกอริทึม



## ❖ Mathematical Analysis

## ❖ Experimental analysis

- ❖ แปลงอัลกอริทึมเป็นโปรแกรม
- ❖ สั่งทำงานกับข้อมูลทดสอบ
- ❖ จับเวลาการทำงาน
- ❖ บันทึกผล
- ❖ วิเคราะห์หาความสัมพันธ์ของเวลาการทำงานกับปริมาณข้อมูล

# Selection Sort

```
selectionSort( d[1..n] ) {  
    for (k = n; k > 1; k--) {  
        maxI = 1  
        for (i = 2; i <= k; i++)  
            if (d[i] > d[maxI]) maxI = i  
        d[k] ↔ d[maxI]  
    }  
}
```

# Selection Sort (ภาษา C)

```
void selectionSort(int d[], int n) {  
    int k, i;  
    for (k = n - 1; k > 0; k--) {  
        int maxI = 1;  
        for (i = 0; i <= k; i++)  
            if (d[i] > d[maxI]) maxI = i;  
        int t = d[k];  
        d[k] = d[maxI];  
        d[maxI] = t;  
    }  
}
```

# เขียนโปรแกรมทดลองจับเวลา

```
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>

int main(int argc, char *argv[]) {
    int i, k, n, repeat = 10;
    for( n = 0; n<=20000; n+=2000) {
        int *d = malloc( n * sizeof(int) );
        long sum = 0;
        for (k=0; k<repeat; k++) {
            for (i=0; i<n; i++) d[i] = i;
            long start = GetTickCount();
            selectionSort(d, n);
            sum += GetTickCount() - start;
        }
        printf("%d \t %f \n", n, (float)sum/repeat);
    }
    system("PAUSE");
}
```

# Selection Sort (ภาษา Java)

```
static void selectionSort(int[] d) {  
    for (int k = d.length-1; k > 0; k--) {  
        int maxI = 1;  
        for (int i = 0; i <= k; i++)  
            if (d[i] > d[maxI]) maxI = i;  
        int t = d[k];  
        d[k] = d[maxI];  
        d[maxI] = t;  
    }  
}
```

# เขียนโปรแกรมทดลองจับเวลา

```
public class TestSelectionSort {  
    public static void main(String[] args) {  
        int repeat = 10;  
        for (int n = 0; n <= 20000; n += 2000) {  
            int[] d = new int[n];  
            long sum = 0;  
            for (int k = 0; k < repeat; k++) {  
                for (int i = 0; i < n; i++) d[i] = i;  
                long start = System.currentTimeMillis();  
                selectionSort(d);  
                sum += System.currentTimeMillis() - start;  
            }  
            System.out.printf("%d \t %f \n",  
                n, (double)sum/repeat);  
        }  
        for (int i = 0; i < 10000; i++)  
            selectionSort(new int[1]);  
    }  
}
```

# สั่งทำงาน + บันทึกผล

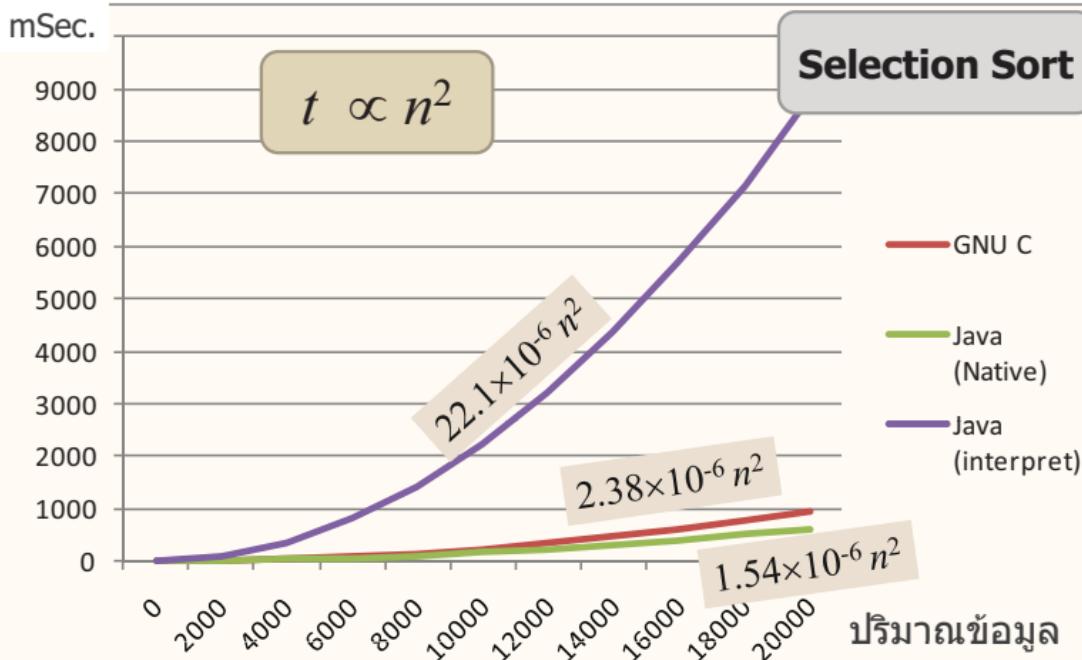
- ❖ ภาษา C : ใช้ GNU-C Compiler (3.4.2)
- ❖ ภาษา Java : ใช้ Java 6 (build 1.6.0\_14-b08)

- ❖ interpreted-only mode  
(option -Xint ตอนทำงาน)
- ❖ Compilation to native code  
(บังคับ compile ด้วยการเรียก  
เมธอดช้า ๆ สัก 10,000 ครั้ง)

**❖ Lenovo X200  
 Intel® Core™ 2 Duo  
 P8400 @ 2.26GHz  
 3GB Ram  
 Windows XP**

n	GNU C	Java (Native)	Java (interpret)
0	0	0.0	0.0
2000	9.4	6.3	90.6
4000	39	25.0	359.4
6000	84.4	54.6	806.2
8000	151.6	98.5	1425.0
10000	237.5	156.2	2231.2
12000	343.7	220.4	3190.6
14000	467.2	300.0	4359.5
16000	610.9	393.7	5675.0
18000	773.5	500.0	7160.6
20000	957.8	614.0	8862.3

# เปรียบเทียบเวลาการทำงาน



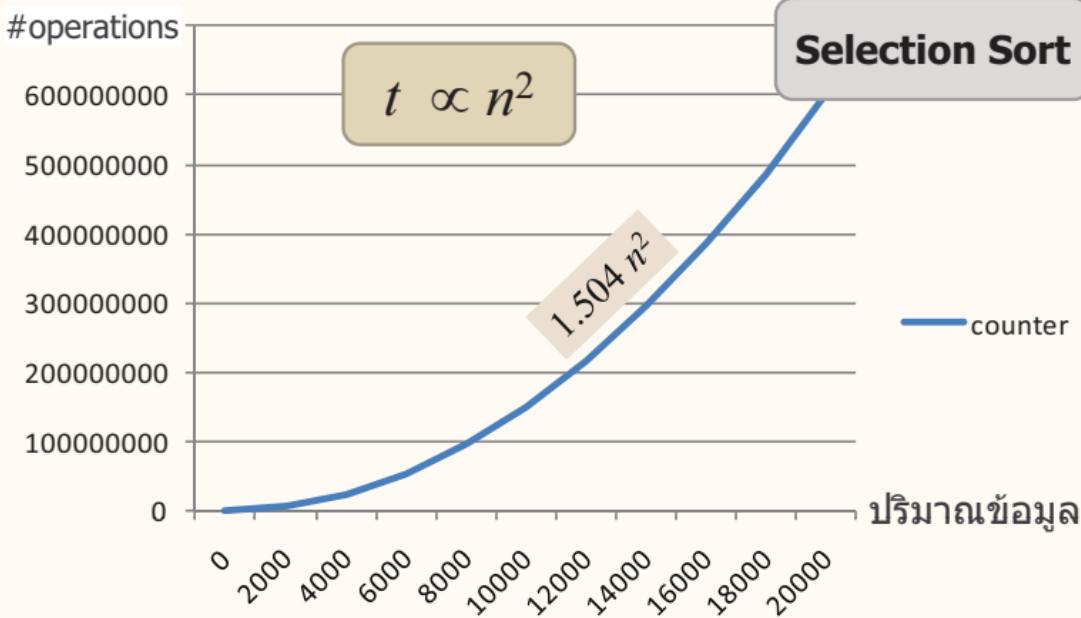
# ใช้จำนวนการทำงานของคำสั่งแทนเวลา

```
static int counter = 0;  
static void selectionSort(int[] d) {  
    counter = 1 + d.length + d.length - 1;  
    for (int k = d.length-1; k > 0; k--) {  
        int maxI = 1;  
        counter += 2 + k+2 + k+1;  
        for (int i = 0; i <= k; i++) {  
            counter += 1;  
            if (d[i] > d[maxI]) {  
                maxI = i; counter += 1;  
            }  
        }  
        counter += 3;  
        int t = d[k]; d[k] = d[maxI]; d[maxI] = t;  
    }  
}
```

# เขียนโปรแกรมทดสอบนับคำสั่ง

```
public class TestSelectionSort {
    public static void main(String[] args) {
        int repeat = 10;
        for (int n = 0; n <= 20000; n += 2000) {
            int[] d = new int[n];
            long sum = 0;
            for (int k = 0; k < repeat; k++) {
                for (int i = 0; i < n; i++) d[i] = i;
                counter = 0;
                selectionSort(d);
                sum += counter;
            }
            System.out.printf("%d \t %f \n",
                n, (double)sum/repeat);
        }
    }
}
```

# นับจำนวนการทำงานของคำสั่ง

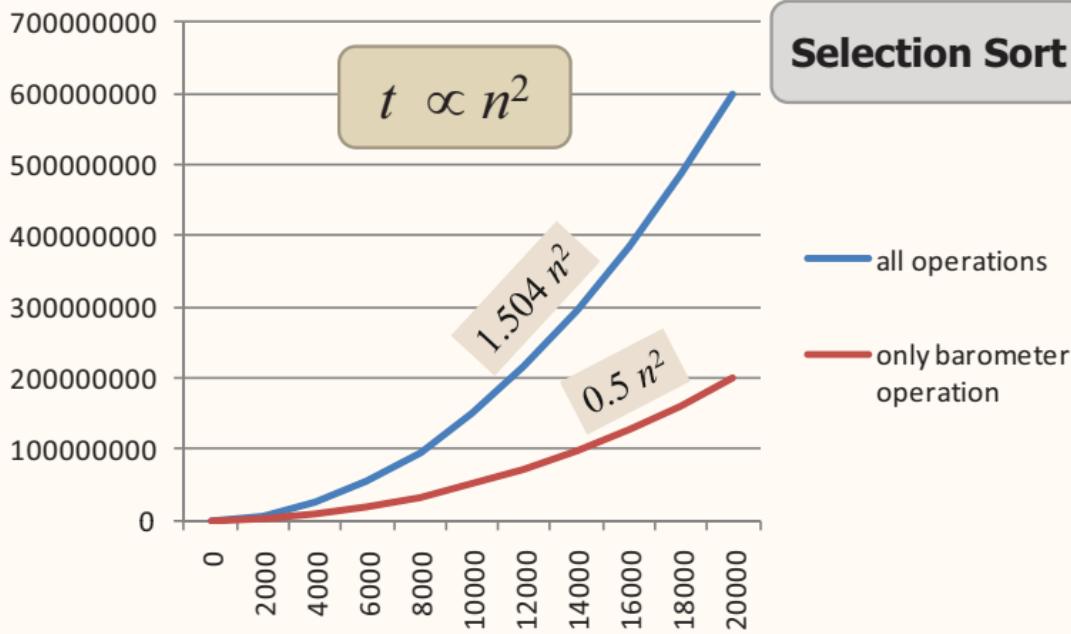


## นับเฉพาะคำสั่งตัวแทน

```
static int counter = 0;

static void selectionSort(int[] d) {
    for (int k = d.length-1; k > 0; k--) {
        int maxI = 1;
        for (int i = 0; i <= k; i++) {
            counter += 1;
            if (d[i] > d[maxI]) {
                maxI = i;
            }
        }
        int t = d[k]; d[k] = d[maxI]; d[maxI] = t;
    }
}
```

# นับทุกคำสั่ง vs. นับคำสั่งตัวแทน



# Mathematical Analysis

- ❖ ไม่ต้องเขียนเป็นโปรแกรม ไม่ต้องสังทำงานจริง  
ไม่ต้องวาดกราฟ วิเคราะห์จากอัลกอริทึม

```
selectionSort( d[1..n] ) {  
    for (k = n; k > 1; k--) {  
        maxI = 1  
        for (i = 2; i <= k; i++)  
            if (d[i] > d[maxI]) maxI = i  
        d[k] ↔ d[maxI]  
    }  
}
```

$$\sum_{k=2}^n \left( \sum_{i=2}^k 1 \right) = \sum_{k=2}^n (k-1) = \sum_{j=1}^{n-1} j = \frac{n(n-1)}{2} = \frac{n^2}{2} - \frac{n}{2}$$

# คำสั่งตัวแทนต้องเป็นคำสั่งพื้นฐาน

## ❖ คำสั่งพื้นฐาน คือคำสั่งที่

- ❖ ใช้เวลาการทำงานไม่เกินค่าคงตัวค่าหนึ่ง
- ❖ ใช้เวลาการทำงานไม่แปรตามขนาดของ input
- ❖ เป็นคำสั่งพื้น ๆ เช่น + - \* / if เปรียบเทียบ return break...

```

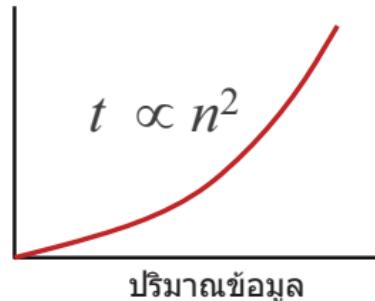
selectionSort( d[1..n] ) {
    for ( k = n; k > 1; k-- ) {
        m = maxI(d, k)
        d[k] ↔ d[m]
    }
}

```

# การวิเคราะห์อัลกอริทึม

- ❖ เวลาการทำงานประมาณจำนวนการทำงานของคำสั่ง
- ❖ จำนวนการทำงานของคำสั่ง ประมาณ  
จำนวนการทำงานของคำสั่งตัวแทน
- ❖ เพื่อความง่าย เรายังคงใช้เวลาด้วย  
การหาความสัมพันธ์ของ
  - ❖ จำนวนการทำงานของคำสั่งตัวแทน
  - ❖ กับปริมาณข้อมูล

จำนวนการทำงาน  
ของคำสั่งตัวแทน  
(เวลาการทำงาน)



# วัดปริมาณข้อมูลขนาดเข้าอย่างไร ?

- ❖ สำหรับ : ถ้าวัดจากขนาด (จำนวนบิต) ของ input
- ❖ ง่ายขึ้น : พิจารณาว่า ขนาดของ input แปรตามค่าอะไรของ input
- ❖ ตัวอย่าง :  $\text{sort}(d[1..n])$ 
  - ❖ input เป็นอาร์เรย์ของจำนวน  $n$  ช่อง
  - ❖ แต่ละจำนวนมีค่าจำกัด (int ใน Java มีค่า  $-2^{31}$  ถึง  $2^{31} - 1$ )
  - ❖ ดังนั้น  $n$  แทนปริมาณข้อมูล
- ❖ ตัวอย่าง :  $\text{shortestPath}(V, E, w, s)$ 
  - ❖ input เป็นกราฟถ่วงน้ำหนัก
  - ❖ ขนาดของกราฟแปรตามจำนวนปมและจำนวนเส้นเชื่อม
  - ❖ ดังนั้น  $|V| + |E|$  แทนปริมาณข้อมูล

# เปรียบเทียบผลการวิเคราะห์

```
for (i = 1; i <= n; i++)
    for (j = i+1; j <= n; j++)
        sum += cosine(d[i][j])*i;
```

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

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

$$\sum_{i=1}^n \sum_{j=1}^n 1 = \sum_{i=1}^n n = n^2$$

แบบใดเร็วกว่า ?

```
for (i = 1; i <= n; i++)
    for (j = 1; j <= n; j++)
        sum += j;
```

# เปรียบเทียบเวลาการทำงาน

```

for (i = 1; i <= n; i++)
    for (j = i+1; j <= n; j++)
        sum += cosine(d[i][j]) * i;
    
```

$\frac{n^2}{2} - \frac{n}{2}$  จำนวนการทำงานของคำสั่ง : น้อยกว่า  
 คำสั่งตัวแทน : ใช้เวลามากกว่า

สรุปลำบากว่าแบบใดเร็วกว่า

$n^2$  จำนวนการทำงานของคำสั่ง : มากกว่า  
 คำสั่งตัวแทน : ใช้เวลาน้อยกว่า

```

for (i = 1; i <= n; i++)
    for (j = 1; j <= n; j++)
        sum += j;
    
```

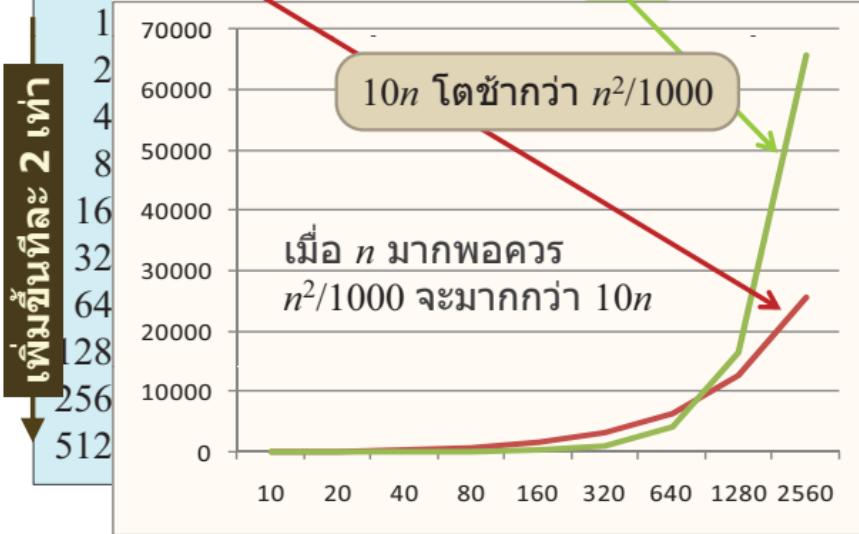
# เปรียบเทียบอัตราการเติบโต

$n$	$n^2$	$\frac{n^2}{2} - \frac{n}{2}$
10	100	45
20	400	190
40	1600	780
80	6400	3160
160	25600	12720
320	102400	51040
640	409600	204480
1280	1638400	818560

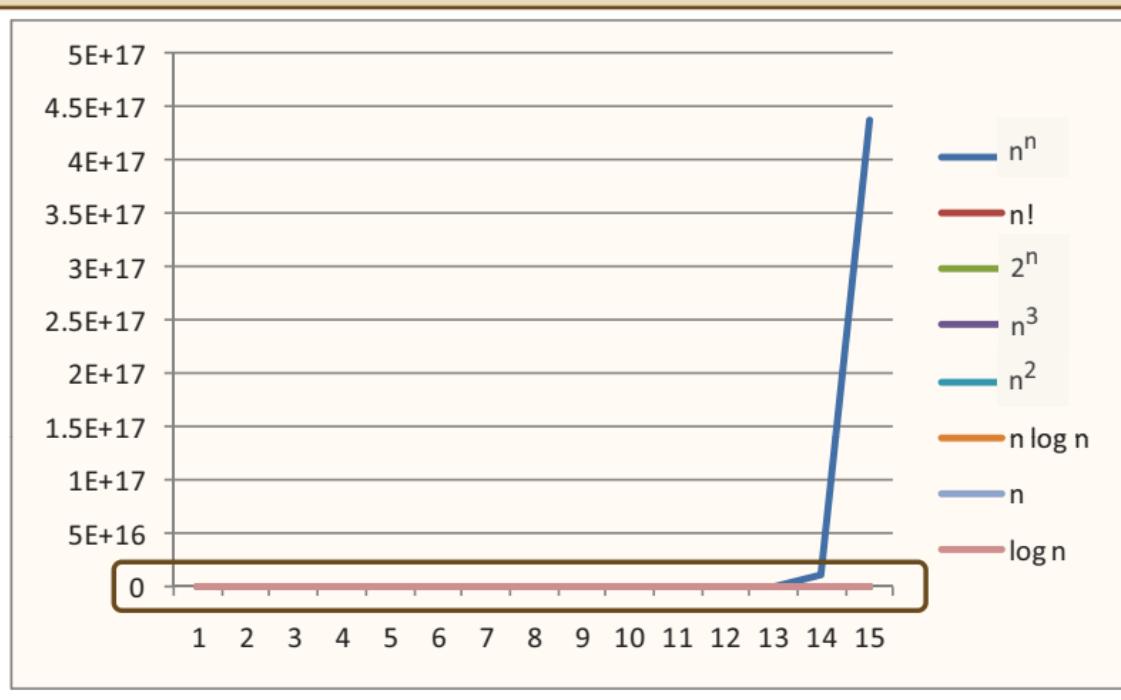
สรุปได้ว่าทั้งคู่  
มีอัตราการเติบโตเท่ากัน

# เปรียบเทียบอัตราการเติบโต

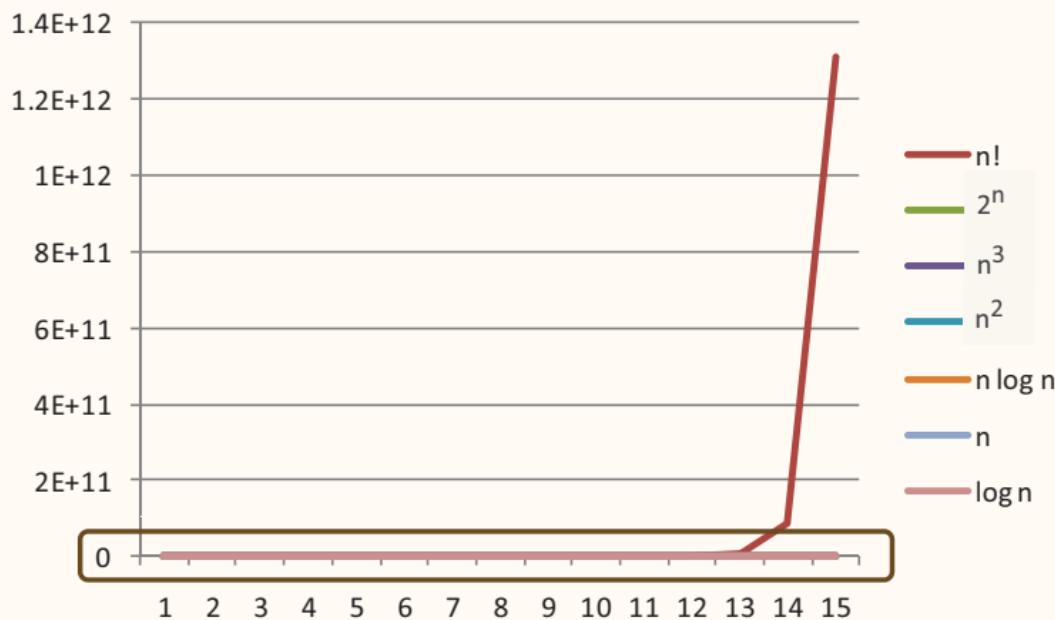
$n$	$10n$	$\frac{n^2}{1000}$		
-----	-------	--------------------	--	--



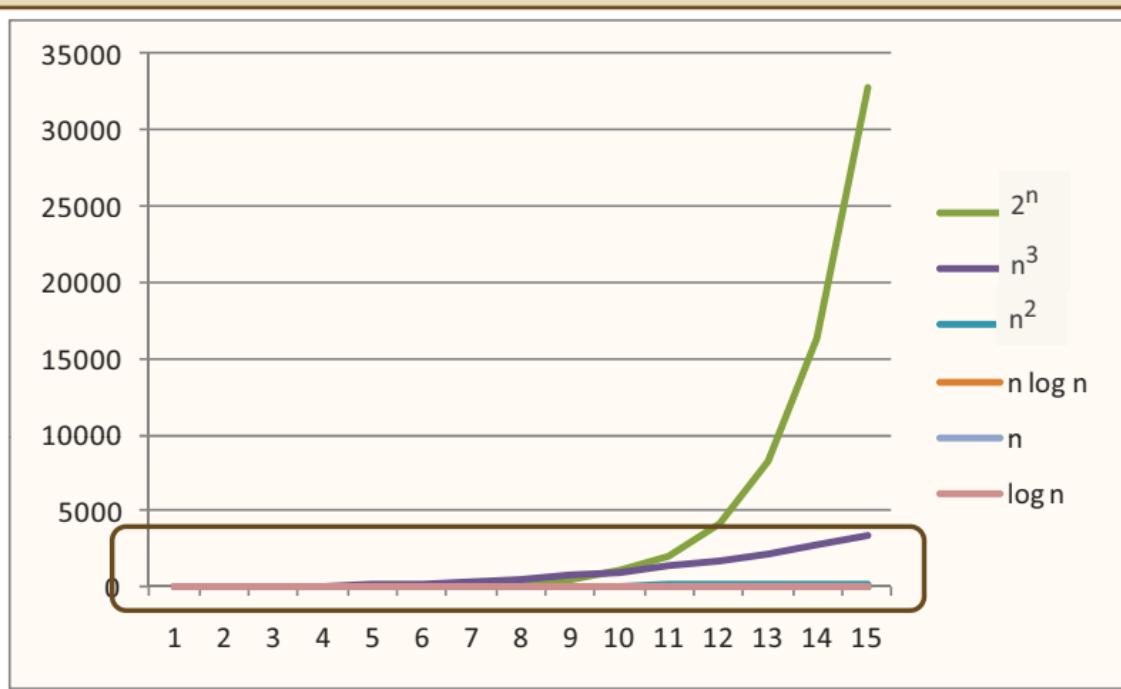
# อัตราการเติบโต



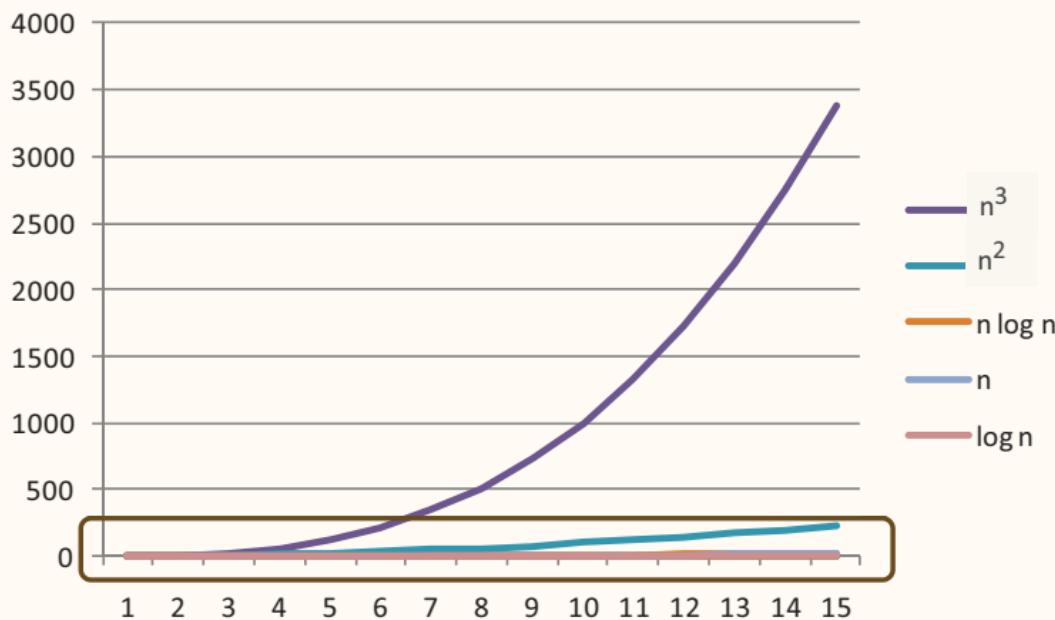
# อัตราการเติบโต



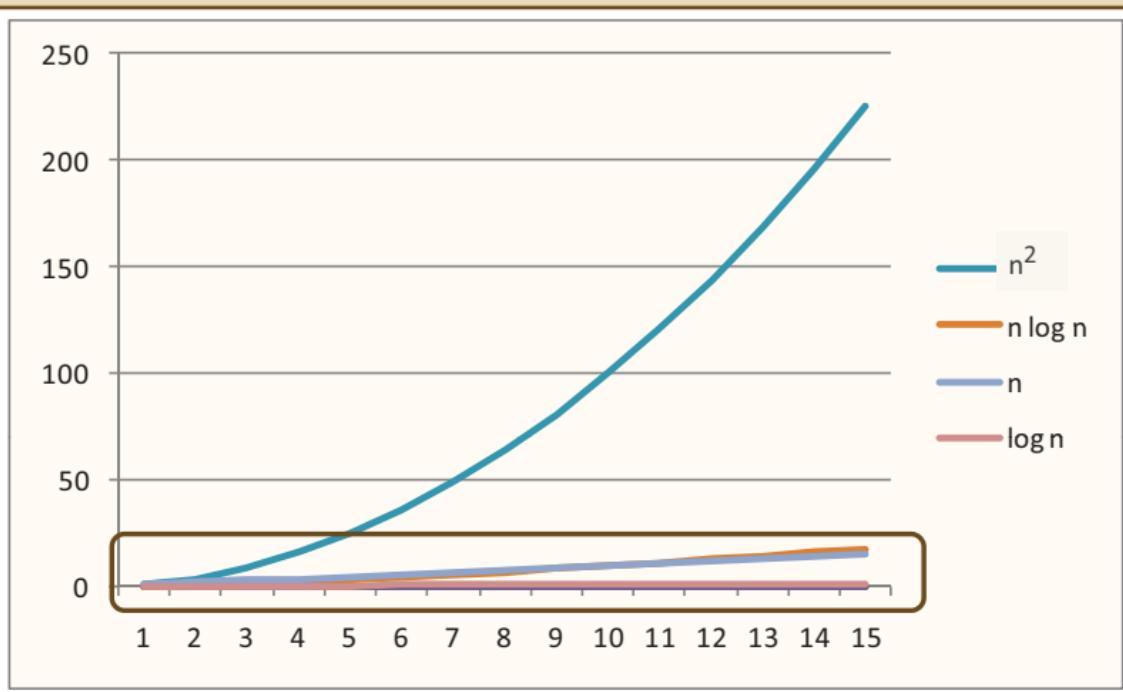
# อัตราการเติบโต



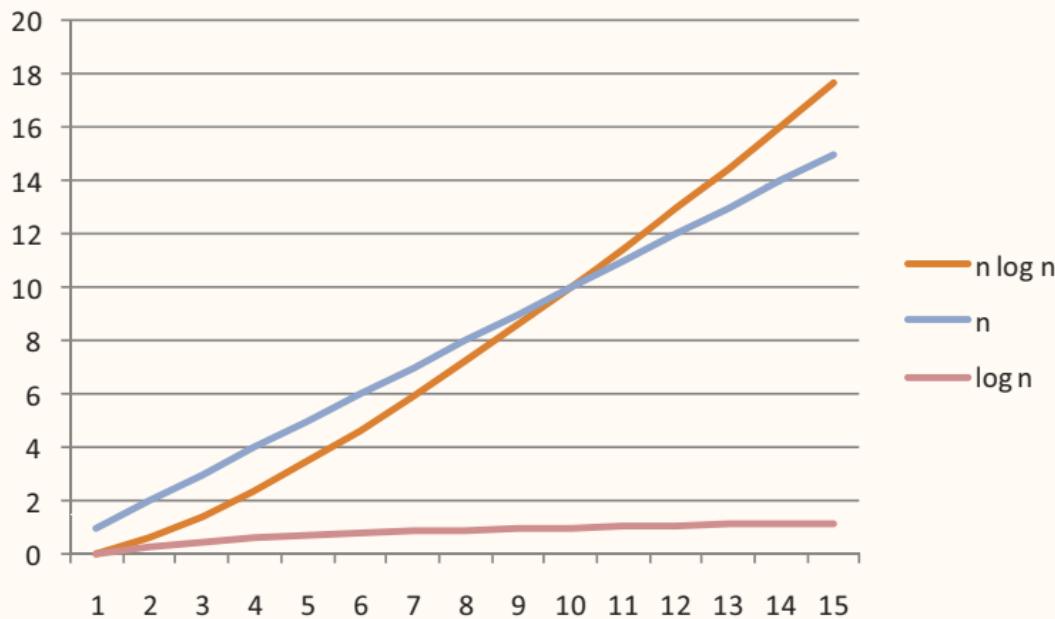
# อัตราการเติบโต



# อัตราการเติบโต



# อัตราการเติบโต



# ทบทวนสูตรคณิตศาสตร์ที่ใช้บ่อย

$$\sum_{k=1}^n k = \frac{n(n+1)}{2}$$

$$S_n = n + (n-1) + \dots + 1$$

$$S_n = 1 + 2 + \dots + n$$

$$2S_n = n(n+1)$$

$$S_n = n(n+1)/2$$

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

# ทบทวนสูตรคณิตศาสตร์ที่ใช้บ่อย

$$\log ab = \log a + \log b$$

$$\log \frac{a}{b} = \log a - \log b$$

$$\log n^a = a \log n$$

$$a^{\log_b n} = n^{\log_b a}$$

$$\log_b a^{\log_b n} = (\log_b n)(\log_b a) = \log_b n^{\log_b a}$$

$$2^{\log_2 n} = n^{\log_2 2} = n$$

## ทบทวนสูตรคณิตศาสตร์ที่ใช้บ่อย

จำนวนเต็ม  $n$  หารด้วย 2 (ปิดเศษทิ้ง) กี่ครั้ง จึงจะเท่ากับ 1

$$31 \rightarrow 15 \rightarrow 7 \rightarrow 3 \rightarrow 1$$

$$32 \rightarrow 16 \rightarrow 8 \rightarrow 4 \rightarrow 2 \rightarrow 1$$

$$\frac{n}{2^k} = 1 \quad 2^k = n, \quad \log_2 2^k = \log_2 n, \quad k = \lfloor \log_2 n \rfloor$$

จำนวนเต็ม  $n$  หารด้วย 3 (ปิดเศษทิ้ง) กี่ครั้ง จึงจะเท่ากับ 1

$$\lfloor \log_3 n \rfloor$$

จำนวนเต็ม  $n$  หารด้วย 4 (ปิดเศษทิ้ง) กี่ครั้ง จึงจะเท่ากับ 1

$$\lfloor \log_4 n \rfloor$$

# ทบทวนสูตรคณิตศาสตร์ที่ใช้บ่อย

$$\sum_{k=0}^n x^k = \frac{x^{n+1} - 1}{x - 1} \quad 2^0 + 2^1 + \dots + 2^n = 2^{n+1} - 1$$

$$\begin{aligned}
 (x-1) \sum_{k=0}^n x^k &= x \sum_{k=0}^n x^k - \sum_{k=0}^n x^k \\
 &= \sum_{k=0}^n x^{k+1} - \sum_{k=0}^n x^k \\
 &= \sum_{j=1}^{n+1} x^j - \sum_{k=0}^n x^k \\
 &= x^{n+1} - 1
 \end{aligned}$$

# ทบทวนสูตรคณิตศาสตร์ที่ใช้บ่อย

$$\sum_{k=0}^{\infty} x^k = \frac{1}{1-x} \quad |x| < 1$$

$$\sum_{k=0}^n x^k = \frac{x^{n+1}-1}{x-1} \quad \sum_{k=0}^{\infty} x^k = \frac{-1}{x-1} = \frac{1}{1-x}$$

ตัวอย่าง  $1 \leq \sum_{k=0}^n 2^{-k} < \sum_{k=0}^{\infty} 2^{-k} = \frac{1}{1-2^{-1}} = \frac{1}{1-\frac{1}{2}} = 2$

$$1 \leq \sum_{k=0}^n 2^{-k} < 2 \quad 1 \leq \sum_{k=0}^n \frac{1}{3^k} < \frac{3}{2}$$

# สรุปอีกครั้ง

$$a^{\log_b n} = n^{\log_b a}$$

$$\sum_{k=0}^n x^k = \frac{x^{n+1} - 1}{x - 1}$$

$$\sum_{k=1}^n k = \frac{n(n+1)}{2}$$

$$\sum_{k=0}^{\infty} x^k = \frac{1}{1-x}$$

$$\log ab = \log a + \log b$$

$$|x| < 1$$

$$\log \frac{a}{b} = \log a - \log b$$

## f(n) vs. g(n)

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \begin{cases} 0 & f(n) \text{ โตช้ากว่า } g(n) \quad f(n) \prec g(n) \\ \infty & f(n) \text{ โตเร็วกว่า } g(n) \quad f(n) \succ g(n) \\ c & f(n) \text{ โตเท่ากับ } g(n) \quad f(n) \asymp g(n) \end{cases}$$

c คือค่าคงตัวที่ไม่ใช่ 0

$$f(n) \preccurlyeq g(n)$$

$$f(n) \succcurlyeq g(n)$$

$f(n)$  โตไม่เร็วกว่า  $g(n)$

$f(n)$  โตไม่ช้ากว่า  $g(n)$

# ตัวอย่าง

$$f(n) = 10n, \quad g(n) = \frac{n^2}{1000}$$

$10n$  โตช้ากว่า  $n^2/1000$

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \lim_{n \rightarrow \infty} \frac{10n}{\left(\frac{n^2}{1000}\right)} = \lim_{n \rightarrow \infty} \frac{10000}{n} = 0$$

$$f(n) = 2n^2 - 5n, \quad g(n) = 10n^2 \quad 2n^2 - 5n \text{ โตเท่ากับ } 10n^2$$

$$\lim_{n \rightarrow \infty} \frac{2n^2 - 5n}{10n^2} = \lim_{n \rightarrow \infty} \left( \frac{2n^2}{10n^2} - \frac{5n}{10n^2} \right) = \lim_{n \rightarrow \infty} \left( \frac{2}{10} - \frac{5}{10n} \right) = \frac{1}{5}$$

## I'Hôpital's Rule

❖ ถ้า  $f(n)$  และ  $g(n)$  เป็นฟังก์ชันที่หาอนุพันธ์ได้ โดยที่

$$\lim_{n \rightarrow \infty} f(n) = \infty, \quad \lim_{n \rightarrow \infty} g(n) = \infty$$

❖ และหาค่าของ  $\lim_{n \rightarrow \infty} \frac{f'(n)}{g'(n)}$  “ได้”

❖ จะได้ว่า  $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \lim_{n \rightarrow \infty} \frac{f'(n)}{g'(n)}$

# ตัวอย่าง

$$f(n) = \log n \quad g(n) = \sqrt{n} \quad \log n \text{ โตข้ากว่า } n^{0.5}$$

$$\lim_{n \rightarrow \infty} \frac{\log n}{\sqrt{n}} = \lim_{n \rightarrow \infty} \frac{\ln n}{\ln 10 \sqrt{n}}$$

สามารถแสดงให้เห็นว่า

$$\begin{aligned}
 &= \frac{1}{\ln 10} \lim_{n \rightarrow \infty} \frac{\ln n}{\sqrt{n}} \\
 &= \frac{1}{\ln 10} \lim_{n \rightarrow \infty} \frac{1/n}{1/(2\sqrt{n})} \\
 &= \frac{1}{\ln 10} \lim_{n \rightarrow \infty} \frac{2}{\sqrt{n}} \\
 &= 0
 \end{aligned}
 \quad (\log n)^{100} \text{ โตข้ากว่า } n^{0.005}$$

## ตัวอย่าง

$$\log n \prec \sqrt{n} \prec n \prec n \log n \prec n^2 \prec n^3 \prec 2^n \prec n^n$$

$$\log n^5 = 5\log n$$

$$n!$$

$$n^2, \quad 10n^2, \quad 2n^2 - 10n, \quad 5n^2 + 8$$

← มีอัตราการเติบโตเท่ากันหมด →

# สัญกรณ์เชิงเส้นกำกับ

```

selectionSort( d[1..n] ) {
    for (k = n; k > 1; k--) {
        maxI = 1
        for (i = 2; i <= k; i++)
            if (d[i] > d[maxI]) maxI = i
        d[k] ↔ d[maxI]
    }
}

```

$$\sum_{k=2}^n \left( \sum_{i=2}^k 1 \right) = \sum_{k=2}^n (k-1) = \sum_{j=1}^{n-1} j = \frac{n(n-1)}{2} = \frac{n^2}{2} - \frac{n}{2}$$

$$= \Theta(n^2)$$

Selection sort ใช้เวลาเป็น  $\Theta(n^2)$

# สัญกรณ์เชิงเส้นกำกับ

- ❖ **little – o**                       $o$
- ❖ **little – omega**                 $\omega$
- ❖ **Big – O**                          $O$
- ❖ **Big – Omega**                   $\Omega$
- ❖ **Big – Theta**                   $\Theta$

Asymptotic Notations

## little - o

$o(g(n)) = \{ f(n) \mid f(n) \prec g(n) \}$  โตช้ากว่า

$$n^{0.98} \in o(n)$$

$$10^6 \in o(n)$$

$$\log n \in o(n)$$

## little - omega

$\omega(g(n)) = \{ f(n) \mid f(n) \succ g(n) \}$  โตเร็วกว่า

$$n^{1.001} \in \omega(n)$$

$$n^2 \in \omega(n)$$

$$2^n \in \omega(n)$$

# Big - Theta

$\Theta(g(n)) = \{ f(n) \mid f(n) \asymp g(n) \}$  โนตเห่ากัน

$$10 \log n \in \Theta(\log n)$$

$$2 + \log n^5 \in \Theta(\log n)$$

$$\ln n \in \Theta(\log n)$$

$$\frac{\log_2 n}{\ln n} = \frac{(\ln n)/(\ln 2)}{\ln n} = \frac{1}{\ln 2}$$

## Big - O

$O(g(n)) = \{ f(n) \mid f(n) \leq g(n) \}$  โอไม่เร็วกว่า

$$32n^{\ln 3} \in O(n^2) \quad \ln 3 < 1.099$$

$$n^2 + 7n \in O(n^2)$$

$$\log n \in O(n^2)$$

# Big - Omega

$\Omega(g(n)) = \{ f(n) \mid f(n) \geq g(n) \}$  โดยไม่จำกัด

$$10n^3 + 5n \in \Omega(n \log n)$$

$$20n \log n + 2n \in \Omega(n \log n)$$

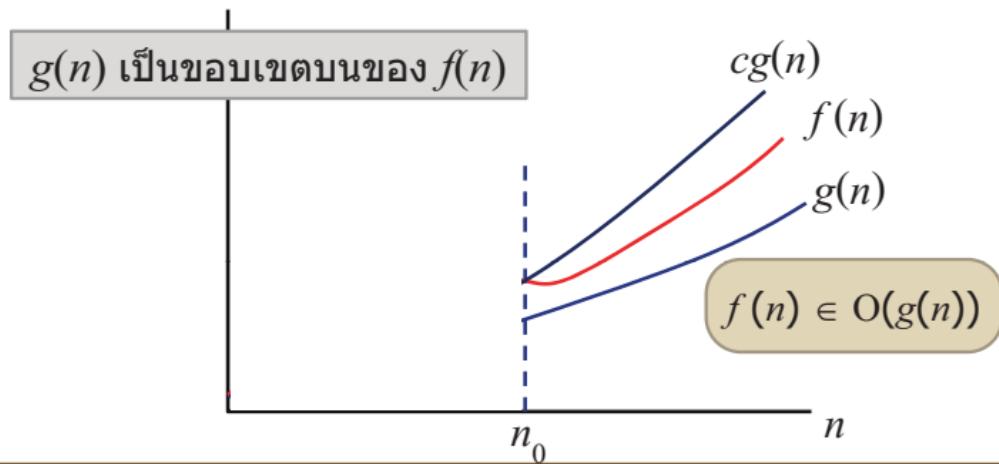
$$n\sqrt{n} \in \Omega(n \log n)$$

# สัญกรณ์เชิงเส้นกำกับ

- ❖  $o(g(n)) = \{ f(n) \mid f(n) \prec g(n) \}$  โตช้ากว่า
- ❖  $\omega(g(n)) = \{ f(n) \mid f(n) \succ g(n) \}$  โตเร็วกว่า
- ❖  $\Theta(g(n)) = \{ f(n) \mid f(n) \asymp g(n) \}$  โตเท่ากัน  
=  $O(g(n)) \cap \Omega(g(n))$
- ❖  $O(g(n)) = \{ f(n) \mid f(n) \leq g(n) \}$  โตไม่เร็วกว่า  
=  $o(g(n)) \cup \Theta(g(n))$
- ❖  $\Omega(g(n)) = \{ f(n) \mid f(n) \geq g(n) \}$  โตไม่ช้ากว่า  
=  $\omega(g(n)) \cup \Theta(g(n))$

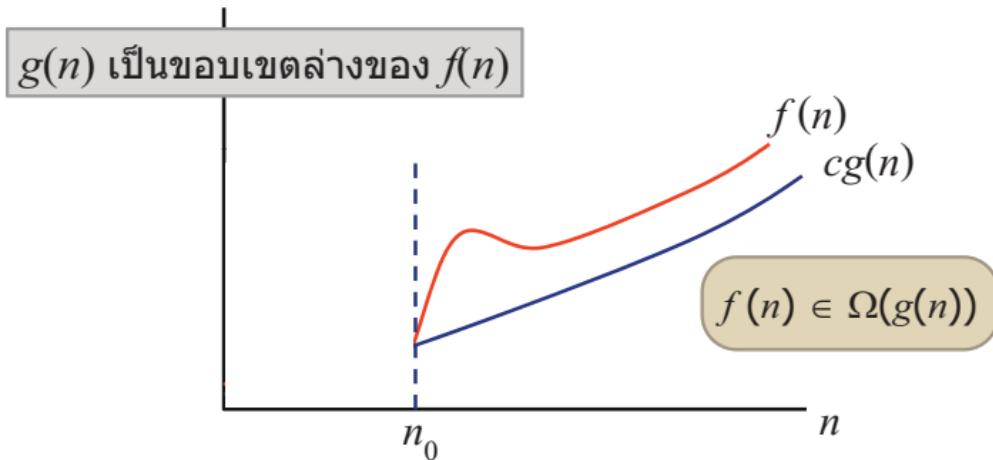
# Big – O : ขอบเขตบน

- ❖  $O(g(n)) = \{ f(n) \mid f(n) \leq g(n) \}$
- ❖  $O(g(n)) = \{ f(n) \mid \text{มีจำนวน } c > 0 \text{ และ } n_0 \geq 0$   
ที่ทำให้  $f(n) \leq cg(n)$  เมื่อ  $n \geq n_0 \}$



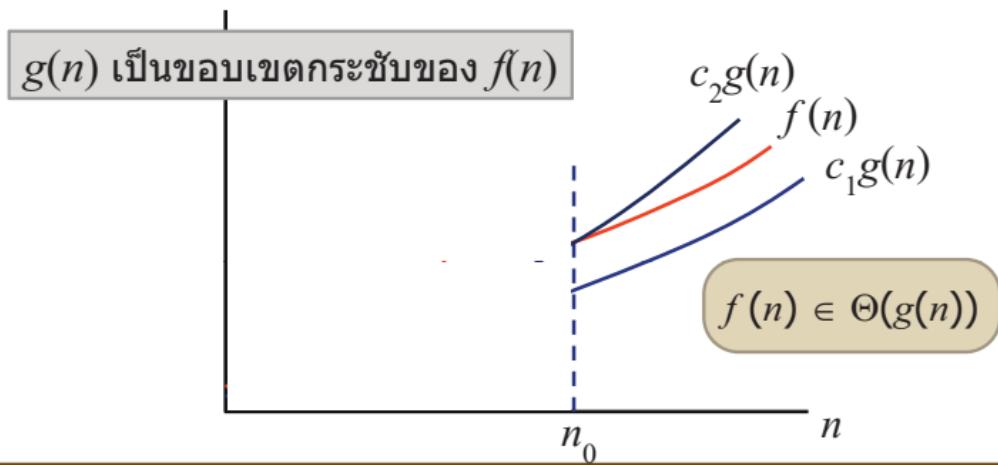
# Big – Omega : ขอบเขตล่าง

- ❖  $\Omega(g(n)) = \{ f(n) \mid f(n) \geq g(n) \}$
- ❖  $\Omega(g(n)) = \{ f(n) \mid \text{มีจำนวน } c > 0 \text{ และ } n_0 \geq 0$   
ที่ทำให้  $f(n) \geq cg(n)$  เมื่อ  $n \geq n_0 \}$



# Big – Theta : ขอบเขตกราฟชี้บ

- ❖  $\Theta(g(n)) = \{ f(n) \mid f(n) \asymp g(n) \}$
- ❖  $\Theta(g(n)) = \{ f(n) \mid \text{มีจำนวน } c_1, c_2 > 0 \text{ และ } n_0 \geq 0$   
ที่ทำให้  $c_1g(n) \leq f(n) \leq c_2g(n)$  เมื่อ  $n \geq n_0$  }



# สัญกรณ์เชิงเส้นกำกับ

❖ เข้าใจพฤติกรรมเมื่อพารามิเตอร์มีค่ามาก

❖  $5n^3 - 2n^2 - n\log n + 7 \in \Theta(n^3)$

❖ วิเคราะห์ได้ง่าย

❖  $\sum_{i=3}^{n-4} \sum_{j=i+2}^n 1 < \sum_{i=1}^n \sum_{j=1}^n 1 = \sum_{i=1}^n n = n^2 = O(n^2)$

❖ จัดกลุ่มฟังก์ชันตามอัตราการเติบโต

❖  $\Theta(1), O(\log n), \Omega(n), \dots$

# วิเคราะห์ง่ายขึ้นด้วย O, Ω, Θ

จะแสดงว่า  $\sum_{i=1}^n \left(\frac{i}{2}\right)^k = \Theta(n^{k+1})$

1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	6	6	6

$$\sum_{i=1}^n \left(\frac{i}{2}\right)^k < \sum_{i=1}^n \left(\frac{n}{2}\right)^k$$

$$< \sum_{i=1}^n n^k$$

$$= n^{k+1}$$

$$\in O(n^{k+1})$$

$$\sum_{i=1}^n \left(\frac{i}{2}\right)^k > \sum_{i=1}^{\lceil n/4 \rceil} \left(\frac{n/2}{2}\right)^k$$

$$= \sum_{i=1}^{\lceil n/4 \rceil} \left(\frac{n}{4}\right)^k$$

$$> \left(\frac{n}{4}\right)^{k+1}$$

$$\in \Omega(n^{k+1})$$

# ตัวอย่าง

จงแสดงว่า  $\sum_{h=0}^{\lfloor \log_2 n \rfloor} \left( \frac{n}{2^h} h \right) = O(n)$

$$\sum_{h=0}^{\lfloor \log_2 n \rfloor} \left( \frac{n}{2^h} h \right) = n \sum_{h=0}^{\lfloor \log_2 n \rfloor} \left( \frac{h}{2^h} \right)$$

$$\sum_{h=0}^{\infty} x^h = 1 + x + x^2 + \dots = \frac{1}{1-x} < n \sum_{h=0}^{\infty} \left( \frac{h}{2^h} \right)$$

$$\sum_{h=0}^{\infty} h x^{h-1} = \frac{1}{(1-x)^2} = 2n \in O(n)$$

$$\sum_{h=0}^{\infty} h x^h = \frac{x}{(1-x)^2} \quad \sum_{h=0}^{\infty} h \left( \frac{1}{2} \right)^h = \frac{1/2}{(1-1/2)^2} = \frac{1/2}{(1/2)^2} = 2$$

# ตัวอย่าง : $\log_2 n!$

$$\begin{aligned} n! &= n \times (n-1) \times \dots \times 2 \times 1 \\ \text{ขอบเขต} &\leq n \times n \times \dots \times n \times n = n^n \\ \text{บน } \log_2 n! &\leq \log_2 n^n = n \log_2 n \text{ เมื่อ } n \geq 1 \end{aligned}$$

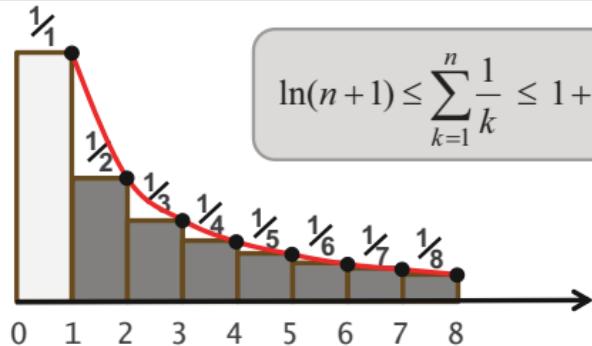
$$\begin{aligned} n! &= n \times (n-1) \times \dots \times (n/2) \times (n/2-1) \times \dots \times 2 \times 1 \\ \text{ขอบเขต} &\geq n/2 \times n/2 \times \dots \times n/2 \times 1 \times \dots \times 1 \times 1 \\ \text{ล่าง} &\geq (n/2)^{n/2} \end{aligned}$$

$$\begin{aligned} \log_2 n! &\geq (n/2) \log_2 (n/2) \\ &= (n/2) \log_2 n - (n/2) \\ &\geq 0.4n \log_2 n \text{ เมื่อ } n \geq 2^5 \end{aligned}$$

$$0.4n \log_2 n \leq \log_2 n! \leq n \log_2 n \text{ เมื่อ } n \geq 2^5$$

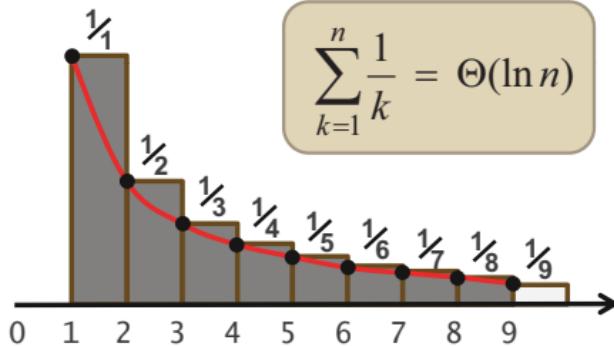
$\log_2 n! \in \Theta(n \log n)$

# Harmonic Number : $H_n = \sum 1/k$



$$\ln(n+1) \leq \sum_{k=1}^n \frac{1}{k} \leq 1 + \ln n$$

$$\begin{aligned} \sum_{k=2}^8 \frac{1}{k} &\leq \int_1^8 \frac{1}{x} dx \\ 1 + \sum_{k=2}^8 \frac{1}{k} &\leq 1 + \int_1^8 \frac{1}{x} dx \\ \sum_{k=1}^8 \frac{1}{k} &\leq 1 + \ln 8 \end{aligned}$$



$$\sum_{k=1}^n \frac{1}{k} = \Theta(\ln n)$$

$$\ln 9 \leq \sum_{k=1}^8 \frac{1}{k} \leq 1 + \ln 8$$

$$\begin{aligned} \sum_{k=1}^8 \frac{1}{k} &\geq \int_1^9 \frac{1}{x} dx \\ \sum_{k=1}^8 \frac{1}{k} &\geq \ln 9 \end{aligned}$$

# การเขียนฟังก์ชันในรูปของ O, Θ และ Ω

❖ ผลบวกของพจน์หลายพจน์ เลือกพจน์ที่ **โตเร็วสุด**

❖ ข้อสังเกต

❖  $cg(n) = \Theta(g(n))$  เมื่อ  $c$  เป็นค่าคงตัว

❖  $\log_a n = \Theta(\log_b n)$  เพราะ  $\log_a n = (\log_a b) \log_b n$

❖  $\cdot \sum \Theta(t(n)) = \Theta\left(\sum t(n)\right)$

❖ เช่น

❖  $a_k n^k + a_{k-1} n^{k-1} + \dots + a_0 = \Theta(n^k)$

❖  $0.001n^3 + 7000n^2 - 11 = \Theta(n^3)$

❖  $\log_2 n^{10} = 10(\log_2 n) = \Theta(\log n)$

❖  $\cdot \sum_{i=1}^n \Theta(i) = \Theta\left(\sum_{i=1}^n i\right) = \Theta\left(n(n+1)/2\right) = \Theta(n^2)$

## อัตราการเติบโต

- ❖ **constant** :  $\Theta(1)$
- ❖ **logarithmic** :  $\Theta(\log n)$
- ❖ **polylogarithmic** :  $\Theta(\log^c n)$ ,  $c \geq 1$
- ❖ **sublinear** :  $\Theta(n^a)$ ,  $0 < a < 1$
- ❖ **linear** :  $\Theta(n)$
- ❖ **quadratic** :  $\Theta(n^2)$
- ❖ **polynomial** :  $\Theta(n^c)$ ,  $c \geq 1$
- ❖ **exponential** :  $\Theta(c^n)$ ,  $c > 1$

# การวิเคราะห์อัลกอริทึม



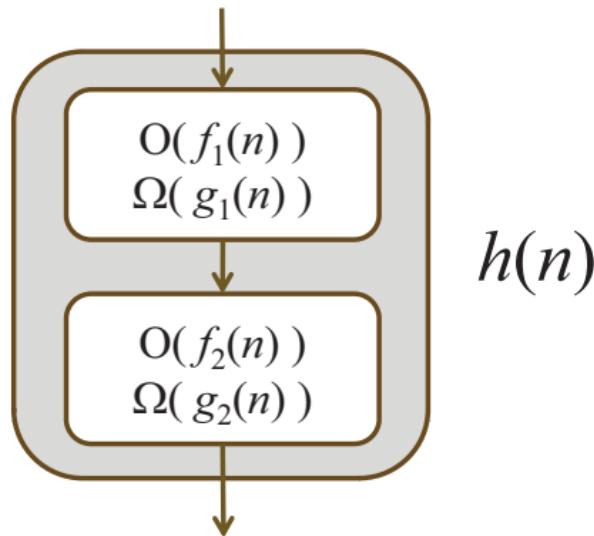
การทำงานแบบลำดับ

การเลือกทำ

การทำงานแบบวงวน

การทำงานแบบเรียกซ้ำ

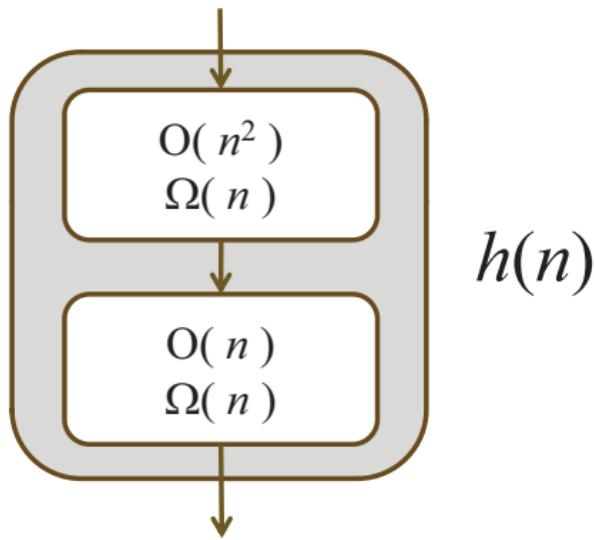
# การทำงานแบบลำดับ



$$h(n) \in O(f_1(n) + f_2(n))$$

$$h(n) \in \Omega(g_1(n) + g_2(n))$$

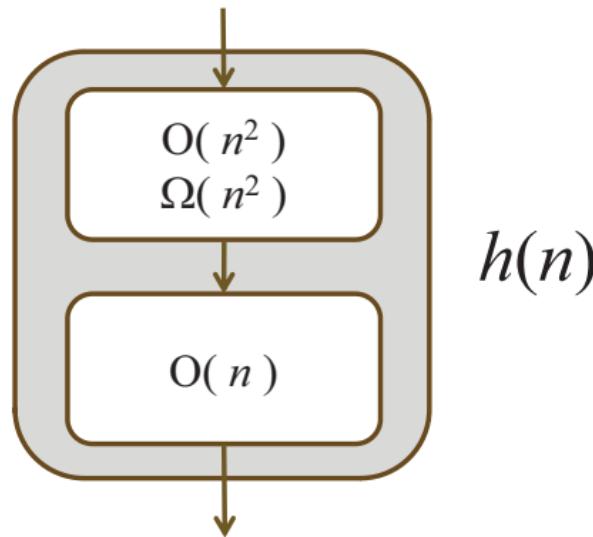
# การทำงานแบบลำดับ



$$h(n) \in O( n^2 + n ) = O( n^2 )$$

$$h(n) \in \Omega( n + n ) = \Omega( n )$$

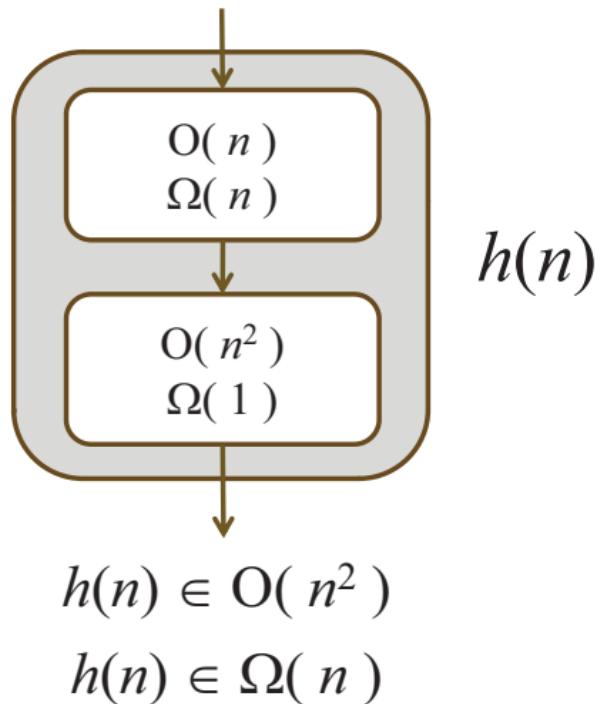
# การทำงานแบบลำดับ



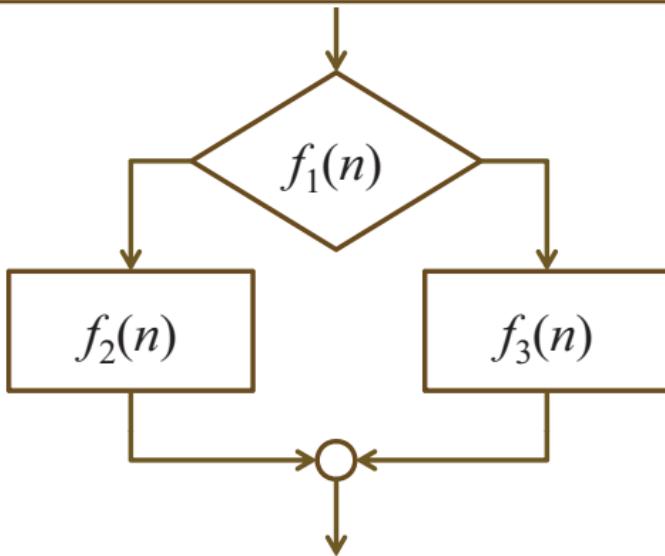
$$h(n) \in O(n^2)$$

$$h(n) \in \Omega(n^2)$$

# การทำงานแบบลำดับ



# การเลือกทำ



$$\max( f_1(n) + f_2(n) , f_1(n) + f_3(n) )$$

## การทำงานแบบบวกวน

```
for (k = 1; k <= n; k++) {  
    P(k)  
}
```

$P(k)$  ใช้เวลา  $t(k)$

$$\sum_{k=1}^n t(k)$$

# ตัวอย่าง : for

```
for (k = 1; k <= n; k++) {  
    process( d, k )  
}
```

**process(d, k)** ใช้เวลา  $\Theta(k^2)$

$$\begin{aligned}\sum_{k=1}^n \Theta(k^2) &= \Theta\left(\sum_{k=1}^n k^2\right) \\ &= \Theta\left(\frac{n(n+1)(2n+1)}{6}\right) \\ &= \Theta(n^3)\end{aligned}$$

# ตัวอย่าง : for

```

for (i = 1; i <= n; i++)
    for (j = 1; j <= n; j++)
        sum += d[i][j]
    
```

$$\begin{aligned}
 \sum_{i=1}^n \sum_{j=1}^n \Theta(1) &= \sum_{i=1}^n \Theta(n) \\
 &= \Theta\left(\sum_{i=1}^n n\right) \\
 &= \Theta(n^2)
 \end{aligned}$$

# ตัวอย่าง : for

```
for (i = 1; i <= n; i++)
    for (j = 1; j <= i; j++)
        sum += d[i][j]
```

$$\begin{aligned} \sum_{i=1}^n \sum_{j=1}^i \Theta(1) &= \sum_{i=1}^n \Theta(i) \\ &= \Theta\left(\sum_{i=1}^n i\right) \\ &= \Theta\left(\frac{n(n+1)}{2}\right) \\ &= \Theta(n^2) \end{aligned}$$

# ตัวอย่าง : for

```
for (i = 2; i <= m-1; i++)
    for (j = 3; j <= i; j++)
        sum += d[i][j]
```

$$\begin{aligned}
 \sum_{i=2}^{m-1} \sum_{j=3}^i \Theta(1) &= \sum_{i=2}^{m-1} \Theta(i) \\
 &= \Theta\left(\sum_{i=2}^{m-1} i\right) \\
 &= \Theta\left(\left(\sum_{i=1}^m i\right) + \Theta(m)\right) \\
 &= \Theta(m^2 + \Theta(m)) = \Theta(m^2)
 \end{aligned}$$

# ตัวอย่าง : Insertion Sort

```

insertionSort( d[1..n] ) {
    for (k = 2; k <= n; k++) {
        t = d[k]
        for (j = k-1; k >= 1; k--) {
            if (t >= d[j]) break
            d[j+1] = d[j]
        }
        d[j+1] = t
    }
}

```

$\Omega(n)$

$$\sum_{k=2}^n O(k)$$


$O(n^2)$

หาที่แทรก  
----- k

เรียงลำดับแล้ว   ข้อมูลที่เหลือ

----->  
ดันมาทางขวา

4, 1, 3, 2, 5

1, 4, 3, 2, 5

1, 3, 4, 2, 5

1, 2, 3, 4, 5

1, 2, 3, 4, 5

# ตัวอย่าง : while

```

log10( n ) {
    c = 0
    while (n > 1) {
        n = n / 3
        c++
    }
    return c
}

```

 $\Theta(\log n)$ 

$$\frac{n}{10^{\log_{10} n}} = 1$$

```

i = 0, j = n
while ( i < j ) {
    i += 4
    j -= 7
}

```

 $\Theta(n)$

# ตัวอย่าง : Binary Search

```
binarySearch( d[1..n] , x ) {  
    left = 1, right = n  
    while ( left <= right ) {  
        mid = ⌊(left + right) / 2⌋  
        if (x = d[mid]) return mid  
        if (x < d[mid])  
            right = mid - 1  
        else  
            left = mid + 1  
    }  
    return -1;  
}
```

$1000$   
 $500$   
 $125$   
 $62$   
 $31$   
 $15$   
 $7$   
 $3$   
 $1$   
 $0$

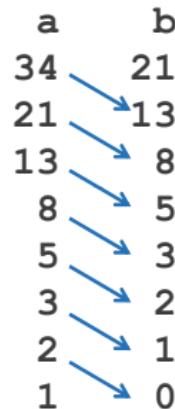
$\frac{n}{2^{\lfloor \log_2 n \rfloor}} = O(\log n)$

right – left + 1 คือจำนวนข้อมูลในช่วงที่สนใจค้น  
ค่าของ right – left + 1 คือ  $n$  ในรอบแรก ลดลง **รอบละครึ่ง**

# ตัวอย่าง : GCD

ถ้า  $a > b$   
 สามารถแสดงว่า  
 $a \text{ mod } b < a/2$   
 เช่น  
 แสดงว่า ทำงาน  
 เป็น  $O(\log n)$

```
gcd(a, b) {
    while (b > 0) {
        t = a mod b
        a = b
        b = t
    }
    return a
}
```



ให้  $a > b$

- ถ้า  $b \leq a/2$ , จะได้  $a \text{ mod } b < a/2$  ( เพราะ  $a \text{ mod } b < b$ )
- ถ้า  $b > a/2$ , จะได้  $a/b < 2$  ดังนั้น  $\lfloor a/b \rfloor = 1$   

$$\begin{aligned} a \text{ mod } b &= a - b \lfloor a/b \rfloor = a - b \\ &< a - a/2 = a/2 \end{aligned}$$

# การทำงานแบบเรียกซ้ำ

```
recursive( ... ) {  
    if ( ... ) return ...  
    ...  
  
    recursive( ... )  
    ...  
    recursive( ... )  
    ...  
}
```

$\Theta(\dots)$

$t(\dots) + \dots + t(\dots)$

$$t(n) = t(\dots) + \dots + t(\dots) + \Theta(\dots)$$

# ตัวอย่าง : การค้นแบบทวิภาค

```
bsearch( d[1..n], x, left, right ) {
    if (left > right) return -1
    mid = ⌊(left + right) / 2⌋
    if (x = d[mid]) return mid
    if (x < d[mid])
        return bsearch( d, x, left, mid - 1 )
    else
        return bsearch( d, x, mid + 1, right )
}
```

ให้  $m = \text{right} - \text{left} + 1$  คือจำนวนข้อมูลในช่วงที่ต้องการค้น  
 ให้  $t(m)$  แทนเวลาการทำงานในการเรียก `bsearch` ที่ช่วงข้อมูล  
 ที่ต้องการค้นมีจำนวน  $m$  ตัว จะได้ว่า

$$t(m) = t(m/2) + \Theta(1), \quad t(0) = \Theta(1)$$

$$t(m) = t(m/2) + \Theta(1)$$

~~$$t(m) = t\left(\frac{m}{2}\right) + \Theta(1)$$~~

~~$$t\left(\frac{m}{2}\right) = t\left(\frac{m}{2^2}\right) + \Theta(1)$$~~

~~$$t\left(\frac{m}{2^2}\right) = t\left(\frac{m}{2^3}\right) + \Theta(1)$$~~

...

~~$$t\left(\frac{m}{2^{k-1}}\right) = t\left(\frac{m}{2^k}\right) + \Theta(1)$$~~

$\Theta(1)$

$$\frac{m}{2^{k-1}} = 1$$

$$m = 2^{k-1}$$

$$k = 1 + \log_2 m$$

$$(2 + \log_2 m)\Theta(1)$$

$$t(m) = \Theta(\log m)$$

กรณีหาไม่พบ

กรณีทั่วไป binary search

$$t(m) = O(\log m)$$

# ตัวอย่าง : Tower of Hanoi

```
hanoi(n, a, b, c) {  
    if (n == 0) return  
    hanoi(n - 1, a, c, b)  
    print("move ", n, a, "->", c)  
    hanoi(n - 1, b, a, c)  
}
```

ให้  $t(n)$  แทนเวลาการทำงานของ `hanoi(n, a, b, c)` จะได้ว่า

$$t(n) = 2t(n - 1) + \Theta(1), \quad t(0) = \Theta(1)$$

$$t(n) = \Theta(2^n)$$

$$t(n) = 2t(n-1) + \Theta(1)$$

$$\begin{aligned} t(n) &= 2t(n-1) + \Theta(1) \\ &= 2(2t(n-2) + \Theta(1)) + \Theta(1) \\ &= 4t(n-2) + 2\Theta(1) + \Theta(1) \\ &= 4(2t(n-3) + \Theta(1)) + 2\Theta(1) + \Theta(1) \\ &= 8t(n-3) + 4\Theta(1) + 2\Theta(1) + \Theta(1) \\ &= 2^3 t(n-3) + 2^2 \Theta(1) + 2^1 \Theta(1) + 2^0 \Theta(1) \\ &\quad \dots \\ &= 2^{\textcolor{red}{n}} t(n-\textcolor{red}{n}) + \dots + 2^2 \Theta(1) + 2^1 \Theta(1) + 2^0 \Theta(1) \\ &= (2^n + \dots + 2^2 + 2^1 + 2^0) \Theta(1) \\ &= (2^{n+1} - 1) \Theta(1) \\ &= \Theta(2^n) \end{aligned}$$

# การะจิง กับ การเรียกซ้ำ

```
DQ( n ) {
    if (n = 0) return
    for (i=1; i<=n; i++)
        for (j=1; j<=n; j++)
            print( i )
    for (i=1; i<=2; i++)
        DQ( n/2 )
}
```

การะจิง

การเรียกซ้ำ

$$t(n) = 2t(n/2) + n^2$$

↑  
แทนการเรียกซ้ำ

แทนการะจิง

# ขนาดข้อมูล กับ การระจิง

$$t(n) = 2t(n/2) + n^2$$

$$n/2$$

การเรียกซ้ำ  
ที่ข้อมูลมี  
ขนาดเล็กลง ๆ

$$n/2$$

$$t(n) = 2t(n/2) + n^2$$

$$(n)^2$$

$$(n/2)^2$$

การระจิง  
สะสม

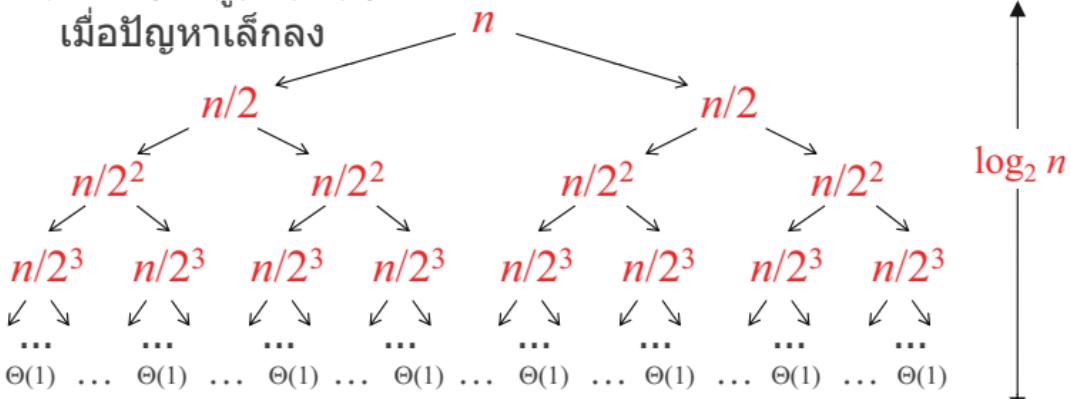
$$(n/2)^2$$

$$t(n/2) = 2t(n/2^2) + (n/2)^2$$

# ขนาดของข้อมูลลดลง ๑

$$t(n) = 2t(n/2) + n^2$$

ขนาดของข้อมูลที่ลดลง  
เมื่อปัญหาเล็กลง



$$t(n/2) = 2t(n/4) + (n/2)^2$$

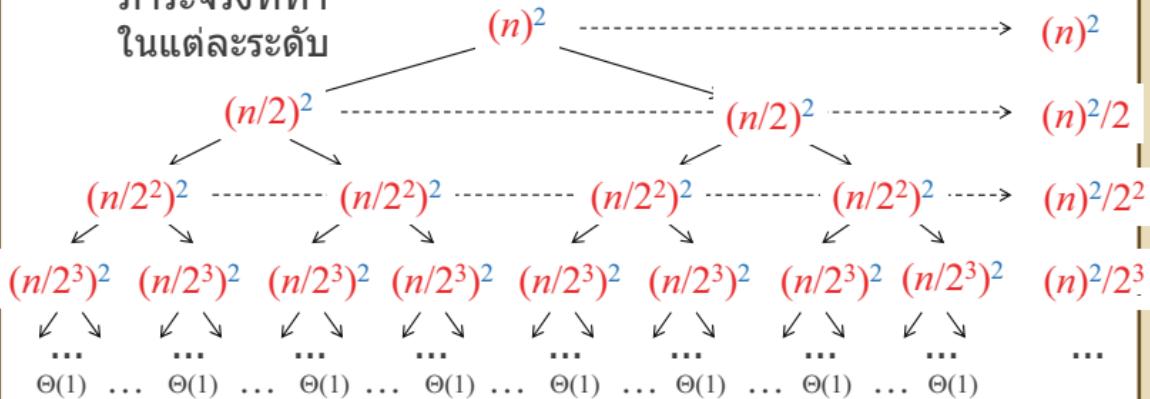
$$t(n/4) = 2t(n/8) + (n/4)^2$$

...

# Recursion Tree : ต้นไม้แสดงการระจิง

$$t(n) = 2t(n/2) + n^2$$

การระจิงที่ทำ  
ในแต่ละระดับ



$$\sum_{k=0}^{\infty} x^k = \frac{1}{1-x}$$

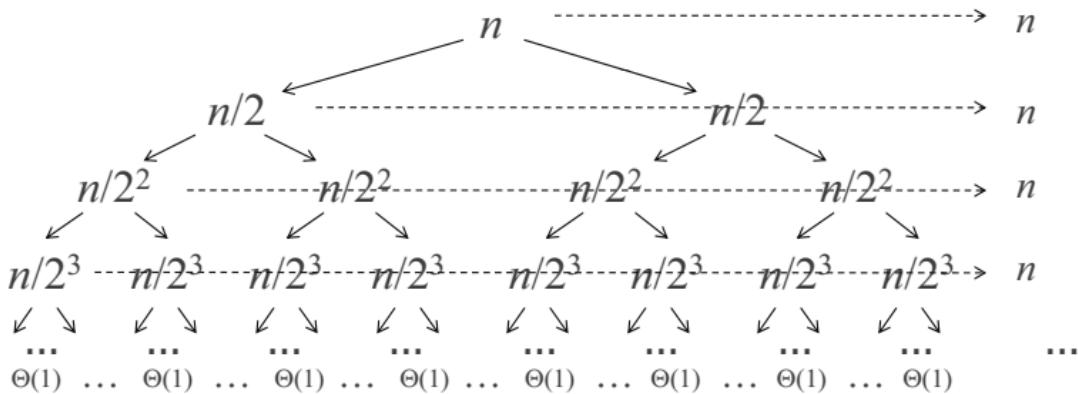
$$\sum_{k=0}^{\log_2 n} \frac{n^2}{2^k} = n^2 \sum_{k=0}^{\log_2 n} \frac{1}{2^k}$$

$$1 \leq < 2$$

$$\Theta(n^2)$$

# Recursion Tree : ตัวอย่าง

$$t(n) = 2t(n/2) + n$$



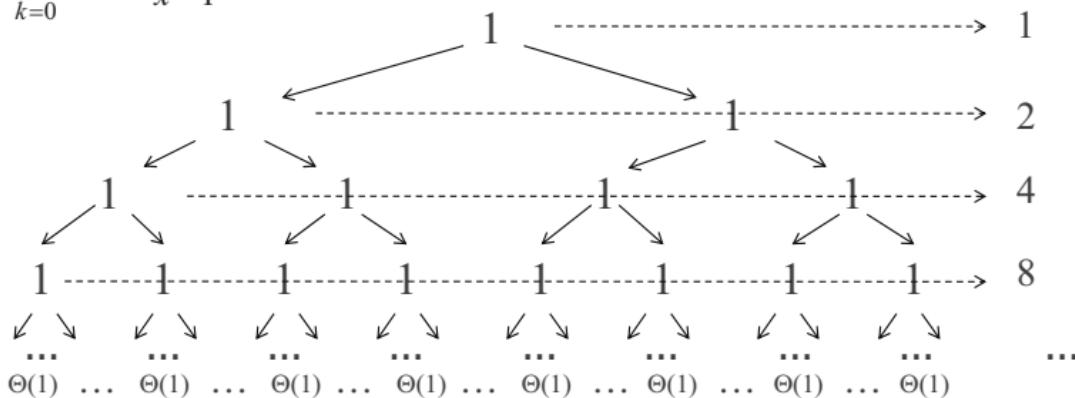
$$\sum_{k=0}^{\log_2 n} n \log_2 n$$

$\Theta( n \log n )$

# Recursion Tree : ตัวอย่าง

$$\sum_{k=0}^n x^k = \frac{x^{n+1} - 1}{x - 1}$$

$$t(n) = 2t(n/2) + 1$$



$$\sum_{k=0}^{\log_2 n} 2^k = 2^0 + 2^1 + \dots + 2^{\log_2 n}$$

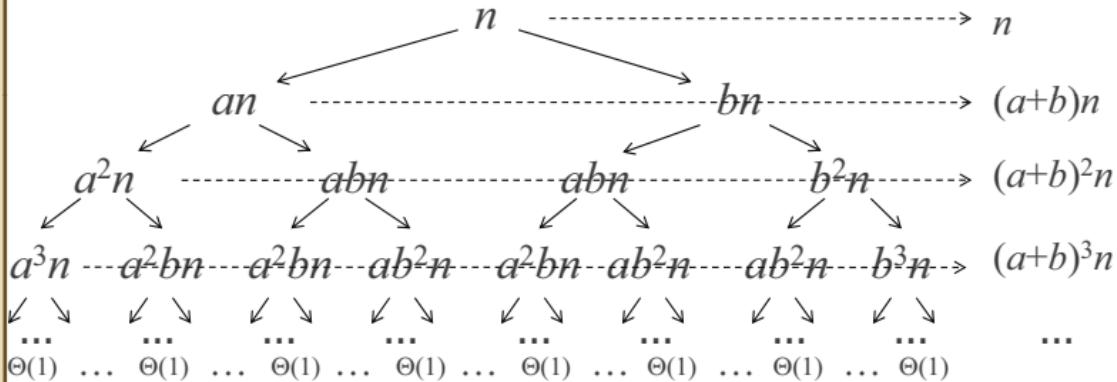
$$= 2^{1+\log_2 n} - 1 = 2 \cdot 2^{\log_2 n} - 1$$

$\Theta(n)$

$$= 2n - 1$$

# Recursion Tree : ตัวอย่าง

$$t(n) = t(an) + t(bn) + n \quad (a+b) < 1$$



$$\sum_{k=0}^{\log n} (a+b)^k n = n \sum_{k=0}^{\log n} (a+b)^k$$

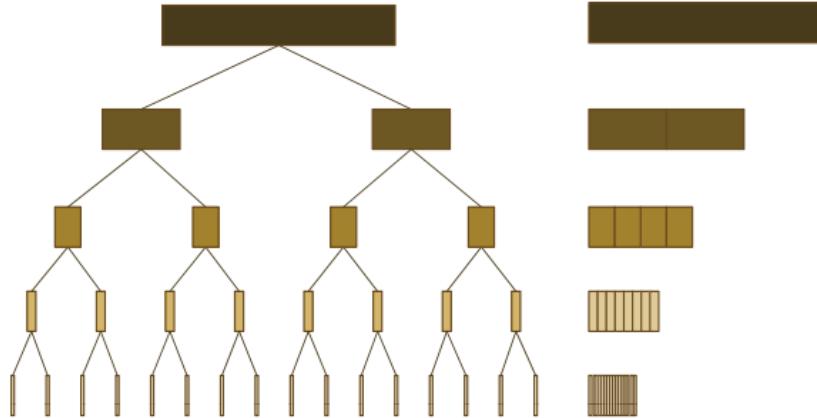
$$\sum_{k=0}^{\infty} x^k = \frac{1}{1-x}$$

$$1 \leq \frac{1}{1-(a+b)} \Theta(n)$$

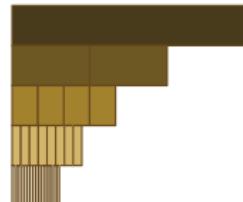
$$T(n) = 2T(n/3) + n$$

3

ภาระจริง  
มากกว่า  
การเรียกข้า

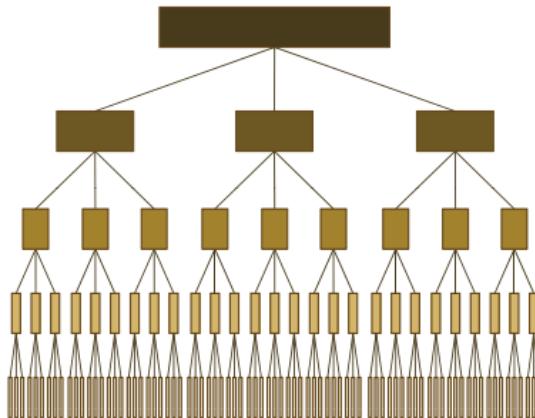


$$n \sum_{k=0}^{\log n} \left(\frac{2}{3}\right)^k < n \left(\frac{1}{1-2/3}\right) = 3n = \Theta(n)$$



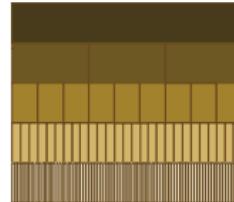
$$T(n) = 3T(n/3) + n$$

2  
การจะริง  
เท่ากับ  
การเรียกซ้ำ



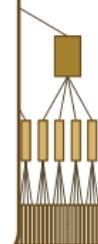
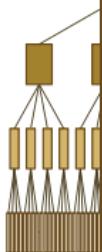
$$\sum_{k=0}^{\log_3 n} n = n \log_3 n$$

$$= \Theta(n \log n)$$



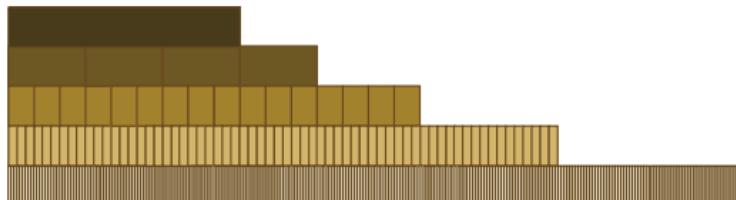
$$T(n) = 4T(n/3) + n$$

$$\begin{aligned}
 n \sum_{k=0}^{\log_3 n} \left(\frac{4}{3}\right)^k &= n \left( \left(\frac{4}{3}\right)^{1+\log_3 n} - 1 \right) = n \left( \frac{4}{3} \left( \frac{4^{\log_3 n}}{3^{\log_3 n}} \right) - 1 \right) \\
 &= n \left( \frac{4}{3} \left( \frac{n^{\log_3 4}}{n} \right) - 1 \right) = \frac{4}{3} n^{\log_3 4} - n \\
 &= \Theta(n^{\log_3 4})
 \end{aligned}$$



1

การะจริง  
น้อยกว่า  
การเรียกช้า



# Master Method (แบบง่าย)

$$t(n) = at(n/b) + \Theta(n^d)$$

$a \geq 1, b > 1,$   
 $d \geq 0$

$$t(n) = \begin{cases} \Theta(n^c) & \text{if } n^d < n^c \\ \Theta(n^c \log n) & \text{if } n^d = n^c \\ \Theta(n^d) & \text{if } n^d > n^c \end{cases}$$

1      2      3

$c = \log_b a$ 
การจะริง
การเรียกซ้ำ

$$t(n) = t(n/2) + \Theta(1) \quad c = \log_2 1 = 0 \quad 1 = n^0 \quad \textcircled{2} \rightarrow \Theta(\log n)$$

$$t(n) = 2t(n/2) + n^2 \quad c = \log_2 2 = 1 \quad n^2 > n^1 \quad \textcircled{3} \rightarrow \Theta(n^2)$$

$$t(n) = 2t(n/2) + n \quad c = \log_2 2 = 1 \quad n = n^1 \quad \textcircled{2} \rightarrow \Theta(n \log n)$$

$$t(n) = 4t(n/2) + 1 \quad c = \log_2 4 = 2 \quad 1 < n^2 \quad \textcircled{1} \rightarrow \Theta(n^2)$$

~~$$t(n) = 2t(n-1) + \Theta(1) \quad t(n) = t(an) + t(bn) + n$$~~

# Master Method

$$t(n) = at(n/b) + \Theta(n^d)$$

$a \geq 1, b > 1$

$$t(n) = \begin{cases} \Theta(n^c) & \text{if } n^d < n^c \\ \Theta(n^c \log n) & \text{if } n^d = n^c \\ \Theta(n^d) & \text{if } n^d > n^c \end{cases}$$

$$c = \log_b a$$

$$t(n) = at(n/b) + f(n)$$

$$t(n) = \begin{cases} \Theta(n^c) & \text{if } f(n) = O(n^{c-\varepsilon}) \quad 1 \quad \varepsilon > 0 \\ \Theta(n^c \log n) & \text{if } f(n) = \Theta(n^c) \quad 2 \\ \Theta(f(n)) & \text{if } f(n) = \Omega(n^{c+\varepsilon}) \quad 3 \\ & a f(n/b) \leq k f(n), k < 1, n \geq n_0 \end{cases}$$

# Master Method : ตัวอย่าง

$$t(n) = at(n/b) + f(n)$$

$$\begin{aligned} a &\geq 1, b > 1, \\ c &= \log_b a \end{aligned}$$

$$t(n) = \begin{cases} \Theta(n^c) & \text{if } f(n) = O(n^{c-\varepsilon}) \quad 1 \quad \varepsilon > 0 \\ \Theta(n^c \log n) & \text{if } f(n) = \Theta(n^c) \quad 2 \\ \Theta(f(n)) & \text{if } f(n) = \Omega(n^{c+\varepsilon}) \quad 3 \end{cases}$$

$a f(n/b) \leq k f(n), k < 1, n \geq n_0$

$$t(n) = 2t(n/2) + 1 \quad c = \log_2 2 = 1, 1 = O(n^{1-\varepsilon}) \quad 1 \rightarrow \Theta(n)$$

$$t(n) = t(n/2) + 1 \quad c = \log_2 1 = 0, 1 = \Theta(n^0) \quad 2 \rightarrow \Theta(\log n)$$

$$t(n) = 3t(n/2) + n^2 \quad c = \log_2 3, n^2 = \Omega(n^{(\log_2 3)+\varepsilon}) \quad \log_2 3 < 1.59$$

$3(n/2)^2 \leq (3/4)n^2, n \geq 0 \quad 3 \rightarrow \Theta(n^2)$

# Master Method : ตัวอย่าง

$$t(n) = at(n/b) + f(n)$$

$$\begin{aligned} a &\geq 1, b > 1, \\ c &= \log_b a \end{aligned}$$

$$t(n) = \begin{cases} \Theta(n^c) & \text{if } f(n) = O(n^{c-\varepsilon}) \quad 1 \quad \varepsilon > 0 \\ \Theta(n^c \log n) & \text{if } f(n) = \Theta(n^c) \quad 2 \\ \Theta(f(n)) & \text{if } f(n) = \Omega(n^{c+\varepsilon}) \quad 3 \\ & af(n/b) \leq kf(n), k < 1, n \geq n_0 \end{cases}$$

$$t(n) = 3t(n/4) + n \log n \quad c = \log_4 3 < 0.793$$

$$n \log n = \Omega(n^{(\log_4 3)+\varepsilon})$$

$$3((n/4) \log (n/4)) \leq (3/4) n \log n, \quad n \geq 0$$

③ →  $\Theta(n \log n)$

# Master Method : ตัวอย่าง

$$t(n) = at(n/b) + f(n) \quad a \geq 1, b > 1, \\ c = \log_b a$$

$$t(n) = \begin{cases} \Theta(n^c) & \text{if } f(n) = O(n^{c-\varepsilon}) \quad 1 \quad \varepsilon > 0 \\ \Theta(n^c \log n) & \text{if } f(n) = \Theta(n^c) \quad 2 \\ \Theta(f(n)) & \text{if } f(n) = \Omega(n^{c+\varepsilon}) \quad 3 \\ & a f(n/b) \leq k f(n), k < 1, n \geq n_0 \end{cases}$$

$$t(n) = 2t(n/2) + n \log n \quad c = \log_2 2 = 1$$

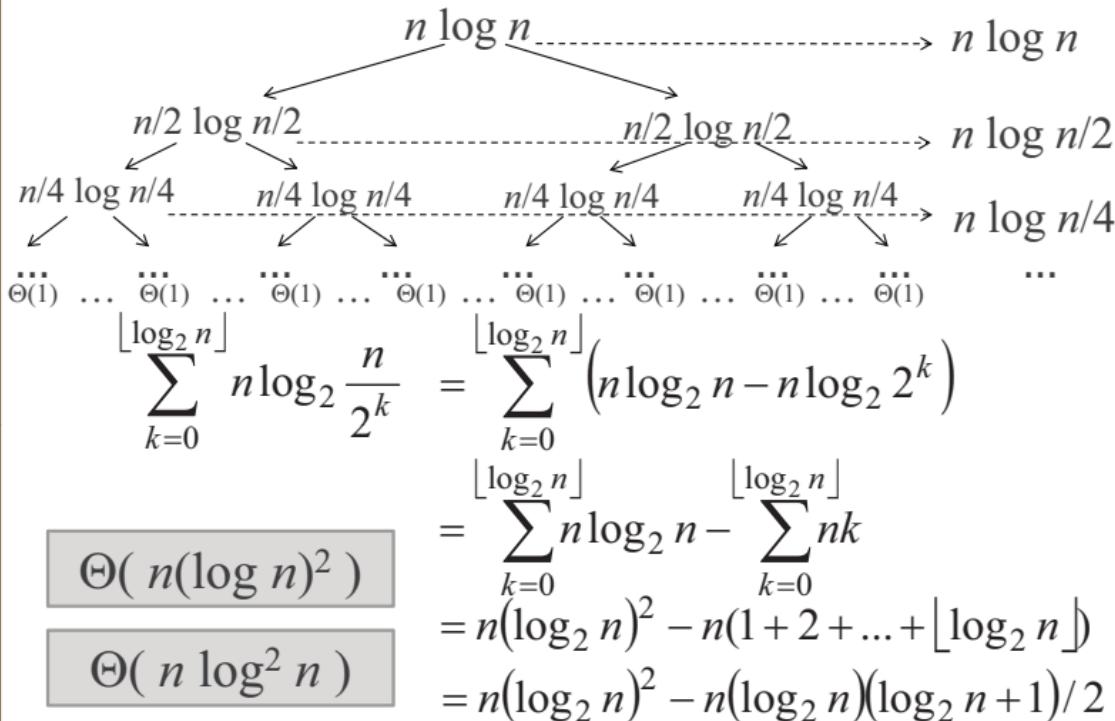
~~$n \log n = O(n^{1-\varepsilon})$~~

~~$n \log n = \Theta(n)$~~

~~$n \log n = \Omega(n^{1+\varepsilon})$~~   $\log n = o(n^\varepsilon)$

ใช้ Master method ไม่ได้  $\rightarrow$  ใช้วิธีอื่น

$$t(n) = 2t(n/2) + n \log n$$



$\Theta(n(\log n)^2)$

$\Theta(n \log^2 n)$

## การวิเคราะห์อัลกอริทึม



### ❖ เวลาการทำงานของอัลกอริทึม ขึ้นกับ

#### ❖ ปริมาณข้อมูลขาเข้า

❖ ช้าเมื่อปริมาณมาก เร็วเมื่อปริมาณน้อย

❖ แต่ก็มีอัลกอริทึม ที่ใช้เวลาคงตัวไม่ขึ้นกับปริมาณ

#### ❖ ลักษณะของข้อมูลขาเข้า

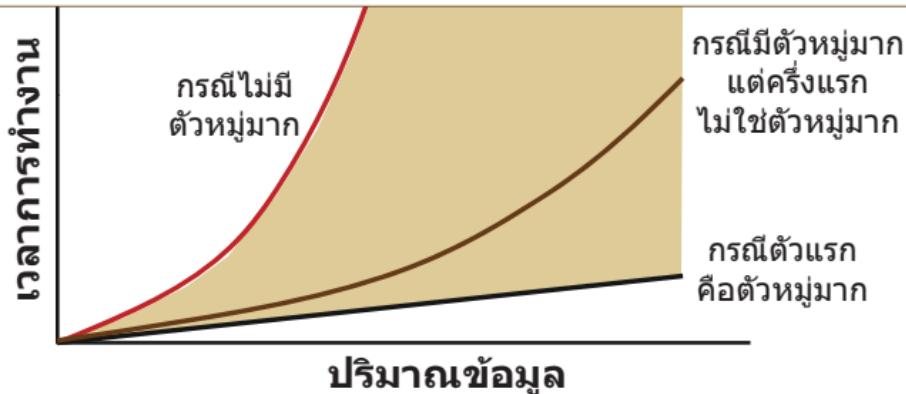
❖ ข้อมูลบางลักษณะ ใช้เวลาทำงานมาก

❖ ข้อมูลบางลักษณะ ใช้เวลาทำงานน้อย

❖ แต่ก็มีอัลกอริทึม ที่ลักษณะข้อมูลไม่มีผลต่อเวลา  
การทำงาน

# การหาตัวหน่วยมาก

```
hasMajority( d[1..n] ) {  
    for (i = 1; i <= n; i++) {  
        c = 0;  
        for (j = 1; j <= n; j++) if (d[i] == d[j]) c++  
        if (c > n/2) return TRUE  
    }  
    return FALSE  
}
```

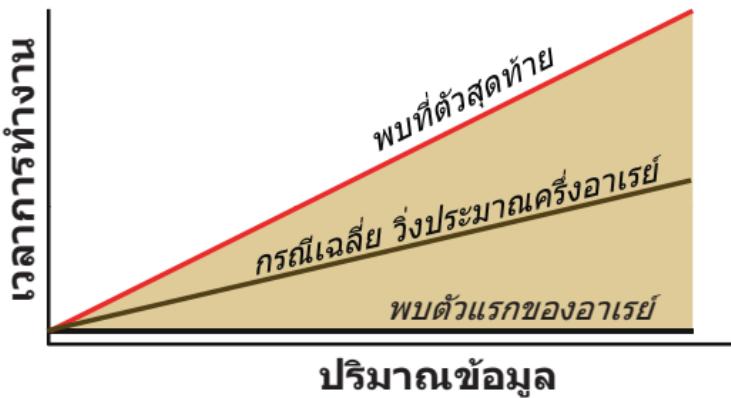


# เวลาการทำงานขึ้นกับลักษณะข้อมูล

```

seqSearch( d[1..n], x ) {
    for (k = 1; k <= n; k++) {
        if (d[k] = x) return k
    }
    return -1
}

```



# การค้นแบบลำดับ

```
seqSearch( d[1..n], x ) {  
    for (k = 1; k <= n; k++) {  
        if (d[k] = x) return k  
    }  
    return -1  
}
```

พบที่  $d[1]$  เปรียบเทียบ 1 ครั้ง

"  $d[2]$  " 2 ครั้ง

... ... ... ...

"  $d[k]$  "  $k$  ครั้ง

มีข้อมูล  $n$  ตัว, ถ้าข้อมูลแต่ละตัวมีโอกาสสุกคันเท่า ๆ กัน  $\rightarrow 1/n$

$$\text{จำนวนการเปรียบเทียบเฉลี่ย} = \sum_{k=1}^n \frac{1}{n} k = \frac{1}{n} \sum_{k=1}^n k = \frac{1}{n} \left( \frac{n(n+1)}{2} \right) = \frac{n+1}{2}$$

# Insertion Sort : Best Case

```

insertionSort( d[1..n] ) {
    for (k = 2; k <= n; k++) {
        t = d[k]
        for (j = k-1; k >= 1; k--) {
            if (t >= d[j]) break
            d[j+1] = d[j]
        }
        d[j+1] = t
    }
}

```

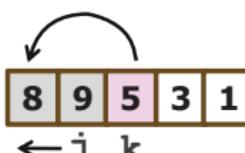
The diagram shows an array of five boxes labeled 1, 3, 5, 8, 9. Below the array, 'j' is positioned under the box containing 5, and 'k' is positioned under the box containing 8. A red box surrounds the condition  $k \geq 1$  in the inner loop of the pseudocode. An arrow points from the value 8 to the index  $j$ , indicating the current element being inserted.

กรณีเร็วสุด : ข้อมูล เรียงลำดับ ทำ  $k \geq 1$  ครั้งเดียว

$$\sum_{k=2}^n 1 = n - 1 = \Theta(n)$$

# Insertion Sort : Worst Case

```
insertionSort( d[1..n] ) {  
    for (k = 2; k <= n; k++) {  
        t = d[k]  
        for (j = k-1; k >= 1; k--) {  
            if (t >= d[j]) break  
            d[j+1] = d[j]  
        }  
        d[j+1] = t  
    }  
}
```



กรณีช้าสุด : ข้อมูล เรียงกลับลำดับ ทำ  $k \geq 1$   $k$  ครั้ง

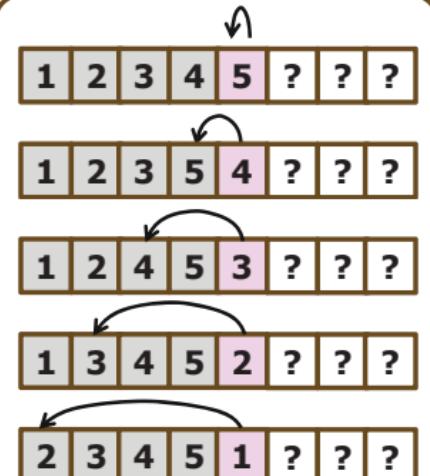
$$\sum_{k=2}^n \sum_{j=0}^{k-1} 1 = \sum_{k=2}^n k = \Theta(n^2)$$

# Insertion Sort : Average Case

```
insertionSort( d[1..n] ) {
    for (k = 2; k <= n; k++) {
        t = d[k]
        for (j = k-1; k >= 1; k--) {
            if (t >= d[j]) break
```

กรณีเฉลี่ย : ทำ  $k \geq 1, 2, \dots, k$  ครั้ง  
ด้วยความน่าจะเป็น  $1/k$  ในแต่ละกรณี

$$\begin{aligned} \sum_{k=2}^n \left( \frac{1}{k} (1+2+\dots+k) \right) &= \sum_{k=2}^n \left( \frac{1}{k} \frac{k(k+1)}{2} \right) \\ &= \frac{1}{2} \sum_{k=2}^n (k+1) = \frac{1}{2} \left( \frac{(n+1)(n+2)}{2} - 3 \right) \\ &= \frac{n^2 + 3n - 4}{4} = \Theta(n^2) \end{aligned}$$



# การหาตัวมากสุด

```
max( d[1..n] ) {  
    max = d[1]  
    for (k = 2; k <= n; k++)  
        if (d[k] > max) max = d[k]  
    return max  
}
```

d เรียงจากน้อยไปมาก  
max เปลี่ยนค่า 0 ครั้ง  
d เรียงจากมากไปน้อย  
max เปลี่ยนค่า 1 ครั้ง

ถ้า d เก็บข้อมูลสุ่ม :

ให้  $C(n)$  แทนจำนวนการเปลี่ยน max เฉลี่ยใน  $d[1..n]$

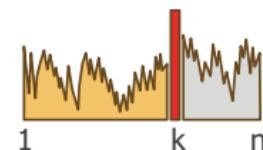
ให้  $C(n,k)$  " " " เมื่อ max คือ  $d[k]$

ถ้า max คือ  $d[k] \rightarrow d[k+1]..d[n]$  ไม่มีผลกับ max

$C(n,k)$  ย่อมเท่ากับ #การเปลี่ยน max เพื่อหา max

ใน  $k-1$  ตัวแรก ตามด้วย การเปลี่ยน max อีกครั้ง

เมื่อเปรียบเทียบกับ  $d[k]$   $C(n,k) = C(k-1) + 1$



$$C(n) = \sum_{k=1}^n \frac{1}{n} C(n, k) = \frac{1}{n} \sum_{k=1}^n (C(k-1) + 1)$$

# การหาค่ามากสุด

$$C(n) = \frac{1}{n} \sum_{k=1}^n (C(k-1) + 1)$$

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

$$nC(n) = n + C(1) + C(2) + \dots + C(n-1)$$

$$(n-1)C(n-1) = (n-1) + C(1) + C(2) + \dots + C(n-2)$$

$$nC(n) - (n-1)C(n-1) = 1 + C(n-1)$$

$$nC(n) - nC(n-1) = 1$$

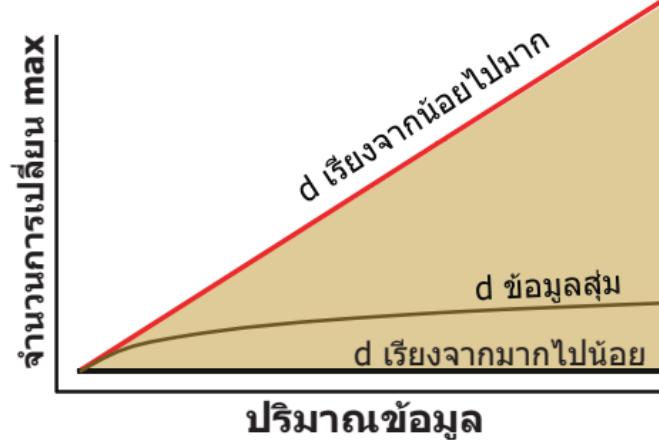
$$C(n) = \frac{1}{n} + C(n-1) = \frac{1}{n} + \frac{1}{n-1} + C(n-2)$$

$$= \frac{1}{n} + \frac{1}{n-1} + \dots + \frac{1}{1} \quad \Theta(\ln n)$$

# การหาค่ามากสุด : การเปลี่ยน max

```
max( d[1..n] ) {  
    max = d[1]  
    for (k = 2; k <= n; k++)  
        if (d[k] > max) max = d[k]  
    return max  
}
```

$\Theta(n)$



$\Theta(\log n)$

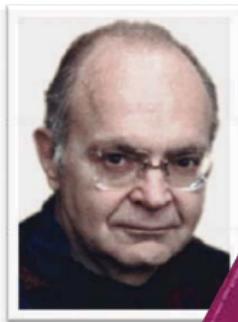
$\Theta(1)$

## สรุป

- ❖ ศึกษาประสิทธิภาพของอัลกอริทึม โดยไม่ต้องเขียนโปรแกรมและสั่งทำงานจริง
- ❖ หาความสัมพันธ์ระหว่างเวลาการทำงานกับปริมาณข้อมูล และ ลักษณะข้อมูล
- ❖ หาอัตราการเติบโตของเวลาการทำงาน
- ❖ แทนเวลาด้วยจำนวนการทำงานของคำสั่งตัวแทน
- ❖ วิเคราะห์ได้ง่ายขึ้นด้วยสัญกรณ์เชิงเส้นกำกับ
- ❖ ใช้ดีในการเปรียบเทียบอัลกอริทึม

# Prof. Donald Knuth

❖ Father of "Analysis of Algorithms"



# อัลกอริทึม แบบแบ่งแยกและเอาชนะ

สมชาย ประสิทธิ์ยุตระกูต

# หัวข้อ

- ❖ โครงของอัลกอริทึมแบบ Divide & Conquer
- ❖ ตัวอย่าง
  - ❖ Binary search
  - ❖ Modular exponentiation
  - ❖ Karatsuba integer multiplication
  - ❖ Strassen matrix multiplication
  - ❖ Merge sort
  - ❖ Quick sort
  - ❖ Quick select
  - ❖ Closest point
  - ❖ Min + Max
  - ❖ Celebrity problem

# Divide – Conquer – Combine

```
DQ( P ) {  
    if ( P is trivial ) return Solve( P )  
    Divide P into P1, P2, ..., Pk  
    for ( i = 1 to k )  
        Si = DQ( Pi )  
    S = Combine( S1, S2, ..., Sk )  
    return S  
}
```

# การค้นแบบทวิภาค (binary search)

- ❖ **input** :  $x$  และ  $D = \langle d_1, d_2, d_3, \dots, d_n \rangle$   
 $d_1 \leq d_2 \leq d_3 \leq \dots \leq d_n$   
 $x$  และ  $d_i$  เป็นจำนวนจริง
- ❖ **output** : ค่า  $k$  ที่  $d_k = x$  ถ้าหาไม่พบ คืน -1

	1	2	3	4	5	6	7	8	9	10	11	12
$d$	2	3	5	9	11	20	25	39	44	49	52	79

$x = 25$

	1	2	3	4	5	6	7	8	9	10	11	12
$d$	2	3	5	9	11	20	25	39	44	49	52	79

$x = 27$

# Binary Search

```

bsearch( d[1..n], x, left, right ) {
    if (left > right) return -1
    mid = ⌊(left + right) / 2⌋
    if (x == d[mid]) return mid
    if (x < d[mid])
        return bsearch( d, x, left, mid - 1)
    else
        return bsearch( d, x, mid + 1, right )
}

```

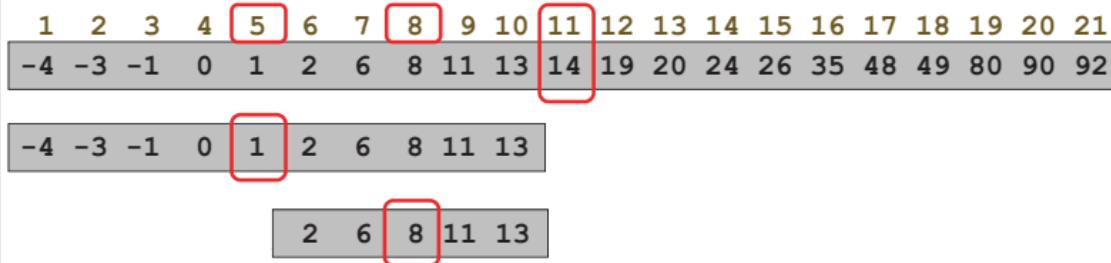
ให้  $m = \text{right} - \text{left} + 1$  คือจำนวนข้อมูลในช่วงที่ต้องการค้น  
 ให้  $t(m)$  แทนเวลาการทำงานในการเรียก bsearch ที่ช่วงข้อมูล  
 ที่ต้องการค้นมีจำนวน  $m$  ตัว จะได้ว่า

$$t(m) = t(m/2) + \Theta(1), \quad t(0) = \Theta(1)$$

$$c = \log_2 1 = 0, 1 = \Theta(m^0) \rightarrow \Theta(\log m)$$

# อย่าลืมใช้ binary search

- ❖ A เป็นอาร์เรย์เก็บข้อมูลแตกต่างกันที่เรียงลำดับแล้ว
- ❖ อยากรู้ว่า ช่องใดใน A ที่  $A[k]$  มีค่าเป็น k



# อย่าลืมใช้ binary search

- ❖ A เป็นอาร์เรย์  $n+1$  ช่อง
- ❖ A เก็บจำนวนเต็มมีค่า 1 ถึง  $n$   
ค่าละอย่างน้อย 1 ช่อง (แสดงว่ามีค่านึงข้า)
- ❖ จงหาว่า ค่าใดซ้ำใน A

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	10	19	15	14	9	13	12	11	8	7	3	4	6	16	17	20	2	5	18	16



1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	16	17	18	19	20
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----	----	----	----	----	----

ลองคิดวิธีที่ใช้ binary search ค้นตัวติดกันเหมือนกัน ?  
ลองคิดวิธีที่ไม่ต้อง sort ข้อมูล ? และหาได้ใน  $O(n)$

# $a^k \bmod m$

- ❖  $a^k \bmod m$  เป็นการคำนวณที่ใช้บ่อยเพื่อเข้ารหัส
- ❖ คำนวณแบบช้า
  - ❖  $a^k \bmod m = a(a^{k-1} \bmod m) \bmod m$
- ❖ คำนวณแบบเร็ว

$$a^k \bmod m = \begin{cases} 1 & k = 0 \\ \left(a^{\lfloor k/2 \rfloor} \bmod m\right)^2 \bmod m & k \text{ is even} \\ a\left(a^{\lfloor k/2 \rfloor} \bmod m\right)^2 \bmod m & k \text{ is odd} \end{cases}$$

$$2^{60} \bmod 10 = (2^{30} \bmod 10)^2 \bmod 10$$

$$2^{30} \bmod 10 = (2^{15} \bmod 10)^2 \bmod 10$$

$$2^{15} \bmod 10 = 2(2^7 \bmod 10)^2 \bmod 10$$

**$a^k \bmod m$** 

$$2^{60} = (2^{30})^2 = 4^2 = 6$$

$$2^{30} = (2^{15})^2 = 8^2 = 4$$

$$2^{15} = 2 \times (2^7)^2 = 2 \times 8^2 = 8$$

$$2^7 = 2 \times (2^3)^2 = 2 \times 8^2 = 8$$

$$2^3 = 2 \times (2^1)^2 = 2 \times 2^2 = 8$$

$$2^1 = 2 \times (2^0)^2 = 2 \times 1^2 = 2$$

$$2^0 = 1$$

$\bmod 10$

## การคำนวณ $a^k \bmod m$

$$a^k \bmod m = \begin{cases} 1 & k = 0 \\ \left(a^{\lfloor k/2 \rfloor} \bmod m\right)^2 \bmod m & k \text{ is even} \\ a \left(a^{\lfloor k/2 \rfloor} \bmod m\right)^2 \bmod m & k \text{ is odd} \end{cases}$$

```
powerMod( a, k, m ) {
    if (k == 0) return 1;
    p = powerMod(a, k / 2, m);
    if ( k is even )
        return (p * p) % m;
    else
        return (a * p * p) % m;
}
```

$$\begin{aligned} t(k) &= t(k/2) + \Theta(1) \\ &= \Theta(\log k) \end{aligned}$$

# การคูณหารเลขขนาดใหญ่ไม่ใช่ $\Theta(1)$

```
powerMod( a, k, m ) {
    if (k == 0) return 1;
    p = powerMod(a, k / 2, m);
    if ( k is even )
        return (p * p) % m;
    else
        return (a * p * p) % m;
}
```

คูณเลข  $\beta$  บิต กับ  $\beta$  บิต ใช้  $\Theta(\beta^2)$

$$\begin{array}{r}
 \times \quad 011 \\
 \quad \quad 110 \\
 \hline
 \quad 000 \\
 \quad 011 \\
 \hline
 011 \\
 \hline
 10010
 \end{array}$$

$k$	#bits
256	8
128	7
64	6
...	...

ถ้า  $a, k$ , และ  $m$  เป็นจำนวนขนาด  $\beta$  บิต

$$\begin{aligned}
 t(\beta) &= t(\beta-1) + \Theta(\beta^2) \\
 &= t(\beta-2) + \Theta(\beta^2) + \Theta(\beta^2) \\
 &= t(\beta-3) + \Theta(\beta^2) + \Theta(\beta^2) + \Theta(\beta^2) \\
 &= \sum_{k=1}^{\beta} \Theta(\beta^2) = \Theta\left(\sum_{k=1}^{\beta} \beta^2\right) = \Theta(\beta^3)
 \end{aligned}$$

# การหาค่า Fibonacci

$$f_n = f_{n-1} + f_{n-2}$$

$$f_0 = 0, f_1 = 1$$

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 370, ...

$$F = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \quad F^2 = \begin{bmatrix} 1 & 1 \\ 1 & 2 \end{bmatrix} \quad F^3 = \begin{bmatrix} 1 & 2 \\ 2 & 3 \end{bmatrix}$$

$$F^4 = \begin{bmatrix} 2 & 3 \\ 3 & 5 \end{bmatrix} \quad F^5 = \begin{bmatrix} 3 & 5 \\ 5 & 8 \end{bmatrix} \quad \text{ท้า } F^n$$

ถ้าการคูณใช้  $\Theta(1)$  :  $T(n) = T(n/2) + \Theta(1) \rightarrow T(n) = \Theta(\log n)$

ถ้าต้องการ  $F_n$  ที่ได้จำนวนขนาดใหญ่ การคูณจะไม่ใช่  $\Theta(1)$  ลองทำดู

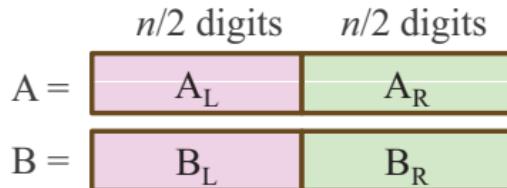
# A × B : คูณได้เร็วกว่า $\Theta(n^2)$ ?

$$\begin{array}{r}
 \times \quad \color{red}{1003} \\
 \color{blue}{0410} \\
 \hline
 \color{black}{0000} \\
 \color{black}{1003} \\
 \color{black}{4012} \\
 \hline
 \color{black}{0000} \\
 \hline
 \color{black}{\underline{\underline{0411230}}}
 \end{array}$$

$$1003 = 10 \times 10^2 + 03, \quad 0410 = 04 \times 10^2 + 10$$

$$\begin{aligned}
 & (10 \times 10^2 + 03) \times (04 \times 10^2 + 10) \\
 &= 10 \times 04 \times 10^4 + (10 \times 10 + 03 \times 04) \times 10^2 + 03 \times 10 \\
 &= 400000 \quad + \quad 11200 \quad \quad \quad + \quad 30 \\
 &= 411230
 \end{aligned}$$

# **A × B**



$$\begin{aligned}
 A \times B &= (A_L 10^{n/2} + A_R) \times (B_L 10^{n/2} + B_R) \\
 &= A_L \times B_L 10^n + (A_L \times B_R + A_R \times B_L) 10^{n/2} + A_R \times B_R
 \end{aligned}$$

$$t(n) = 4t(n/2) + \Theta(n)$$

Master method :  $c = \log_2 4 = 2$ ,  
 $n = O(n^{2-\varepsilon})$  **①**  $\rightarrow t(n) = \Theta(n^2)$



# A x B (อีกแบบ)

$$1003 = 10 \times 10^2 + 03, \quad 0410 = 04 \times 10^2 + 10$$

$$10 \times 04 = 40$$

$$03 \times 10 = 30$$

$$(10 + 03) \times (04 + 10) = 13 \times 14 = 182$$

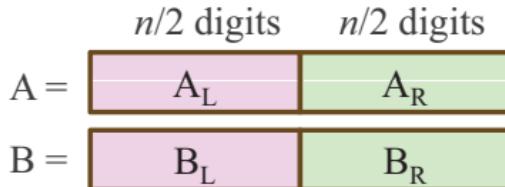
$$(10 \times 10^2 + 03) \times (04 \times 10^2 + 10)$$

$$= 40 \times 10^4 + (182) - 40 - 30 \times 10^2 + 30$$

$$= 400000 + 11200 + 30$$

$$= 411230$$

# A x B : Karatsuba (1962)



$$A \times B = A_L \times B_L 10^n + (A_L \times B_R + A_R \times B_L) 10^{n/2} + A_R \times B_R$$

$$C_1 = A_L \times B_L$$

$$C_2 = A_R \times B_R$$

$$C_3 = (A_L + A_R) \times (B_L + B_R) = (A_L B_L + A_L B_R + A_R B_L + A_R B_R)$$

$$A \times B = C_1 10^n + (C_3 - C_1 - C_2) 10^{n/2} + C_2$$

$$t(n) = 3t(n/2) + \Theta(n)$$

Master method :  $c = \log_2 3$

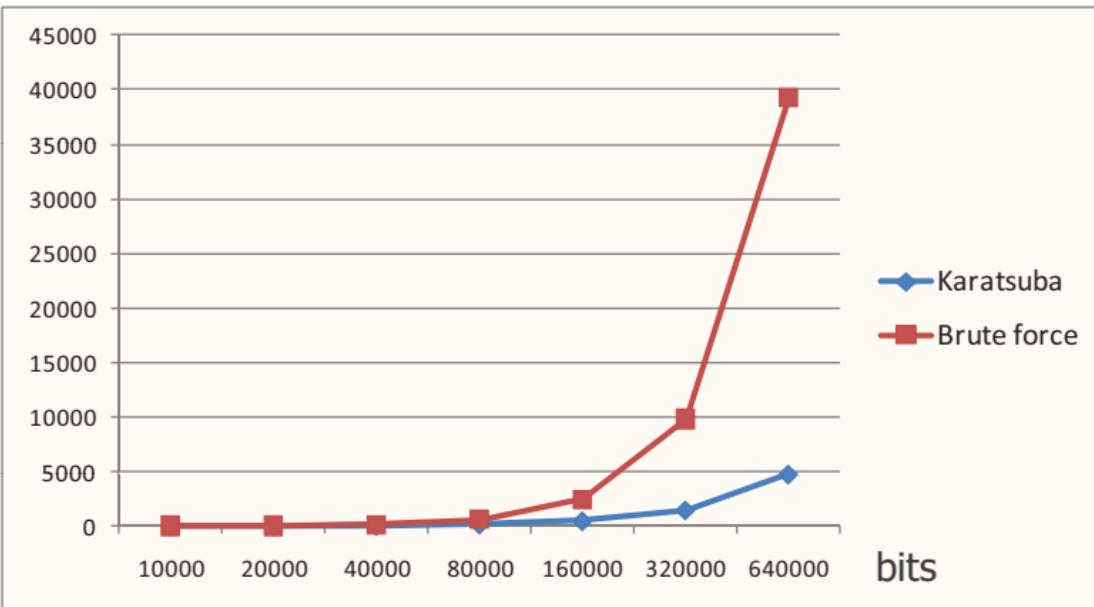
$$n = O(n^{c-\epsilon}) \quad \text{①} \rightarrow t(n) = \Theta(n^{\log_2 3})$$



$$O(n^{1.59})$$

# เปรียบเทียบเวลาการทำงาน

mSec. (interpreted-only mode)



# A x B : การคูณเมทริกซ์

```
MatrixMult( A[1..n][1..n], B[1..n][1..n] ) {  
    C = new array[1..n][1..n]  
    for ( i = 1; i <= n; i++ ) {  
        for ( j = 1; j <= n; j++ ) {  
            C[i][j] = 0  
            for ( k = 1; k <= n; k++ ) {  
                C[i][j] += A[i][k] * B[k][j]  
            }  
        }  
    }  
    return C  
}
```

$$C_{i,j} = \sum_{k=1}^n a_{i,k} \cdot b_{k,j}$$

คูณเมทริกซ์ขนาด  $n \times n$  ส่องตัว ใช้จำนวนการคูณสเกลาร์เป็น  $n^3$

ใช้เวลา  $\Theta(n^3)$

# A x B : Divide & Conquer

$$A = \begin{bmatrix} a_{1,1} & \cdots & a_{1,n/2} & & \cdots & a_{1,n} \\ \vdots & & A_{1,1} & \vdots & A_{1,2} & \vdots \\ a_{n/2,1} & \cdots & a_{n/2,n/2} & & \cdots & a_{n/2,n} \\ \hline & & & & & \\ \vdots & & A_{2,1} & \vdots & A_{2,2} & \vdots \\ a_{n,1} & \cdots & a_{n,n/2} & & \cdots & a_{n,n} \end{bmatrix}$$

## A x B : Divide & Conquer

$$\begin{bmatrix} C_{1,1} & C_{1,2} \\ C_{2,1} & C_{2,2} \end{bmatrix} = \begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix} \times \begin{bmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{bmatrix}$$

$$\begin{aligned} C_{1,1} &= A_{1,1} \times B_{1,1} + A_{1,2} \times B_{2,1} \\ C_{2,1} &= A_{2,1} \times B_{1,1} + A_{2,2} \times B_{2,1} \\ C_{1,2} &= A_{1,1} \times B_{1,2} + A_{1,2} \times B_{2,2} \\ C_{2,2} &= A_{2,1} \times B_{1,2} + A_{2,2} \times B_{2,2} \end{aligned}$$

$$t(n) = 8t(n/2) + \Theta(n^2)$$

Master method :  $c = \log_2 8 = 3$

$$n^2 = O(n^{3-\epsilon}) \quad \textcircled{1} \rightarrow \Theta(n^3)$$



# A x B : Strassen (1969)

$$\begin{bmatrix} C_{1,1} & C_{1,2} \\ C_{2,1} & C_{2,2} \end{bmatrix} = \begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix} \times \begin{bmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{bmatrix}$$

$$M_1 = (A_{12} - A_{22}) \times (B_{21} + B_{22})$$

$$M_2 = (A_{11} + A_{22}) \times (B_{11} + B_{22})$$

$$M_3 = (A_{11} - A_{21}) \times (B_{11} + B_{12})$$

$$M_4 = (A_{11} + A_{12}) \times B_{22}$$

$$M_5 = A_{11} \times (B_{12} - B_{22})$$

$$M_6 = A_{22} \times (B_{21} - B_{11})$$

$$M_7 = (A_{21} + A_{22}) \times B_{11}$$

$$C_{11} = M_1 + M_2 - M_4 + M_6$$

$$C_{12} = M_4 + M_5$$

$$C_{21} = M_8 + M_7$$

$$C_{22} = M_2 - M_3 + M_5 - M_7$$

$$t(n) = 7t(n/2) + \Theta(n^2)$$



Master method :  $c = \log_2 7$

$$n^2 = O(n^{c-\varepsilon}) \quad t(n) = \Theta(n^{\log_2 7})$$

1969 :  $O(n^{2.81})$  Strassen

1987 :  $O(n^{2.376})$  Coppersmith–Winograd

# Master Method (ตีความ)

$$t(n) = \begin{cases} \Theta(n^c) & \text{if } f(n) = O(n^{c-\varepsilon}) \quad 1 \quad \varepsilon > 0 \\ \Theta(n^c \log n) & \text{if } f(n) = \Theta(n^c) \quad 2 \\ \Theta(f(n)) & \text{if } f(n) = \Omega(n^{c+\varepsilon}) \quad 3 \\ af(n/b) \leq kf(n), k < 1, n \geq n_0 \end{cases}$$

$$t(n) = at(n/b) + f(n)$$

ภาระในเรียก recursive  
เพื่อแก้ปัญหาอย่างย่อ

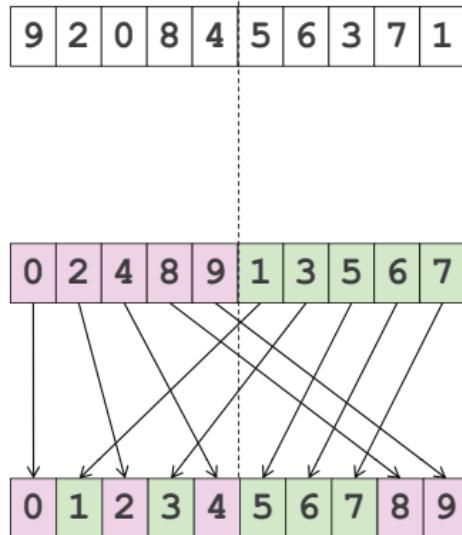
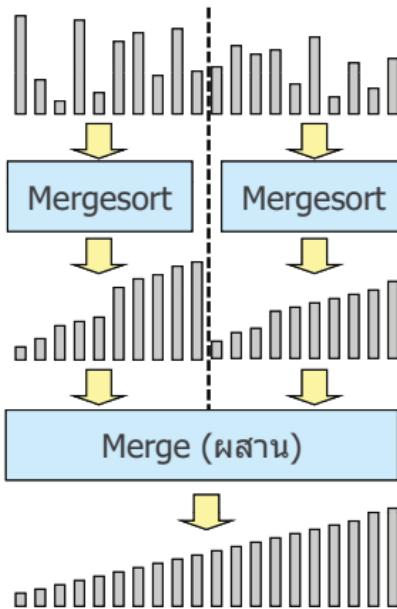
$$\Theta(n^{\log_b a})$$

>  
1  
=  
2  
<  
3

ภาระในการแบ่งปัญหา  
และการรวมคำตอบ

$$f(n)$$

# Merge Sort



# Merge Sort

```
mergeSort( d[1..n], left, right ) {  
    if (left >= right) return;  
    mid = ⌊(left + right) / 2⌋;  
    mergeSort(d, left, mid);  
    mergeSort(d, mid + 1, right);  
    merge(d, left, mid, right);  
}
```

ให้  $t(n)$  คือเวลาในการ mergesort ข้อมูลจำนวน  $n$  ตัว

$$t(n) = 2t(n/2) + (\text{เวลาในการ merge})$$

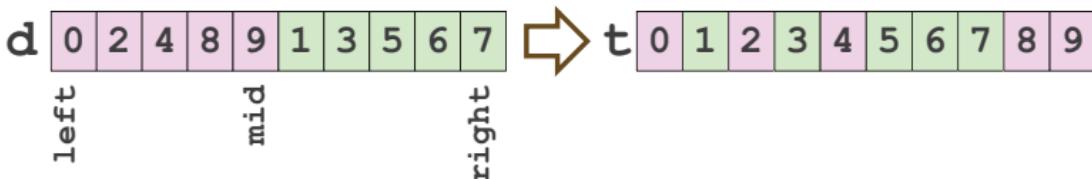
# Merge : #cmps

```

merge( d[1..n], left, mid, right ) {
    create t[left..right]
    i = left, j = mid+1;
    for (k = left; k <= right; k++) {
        if (i > mid) {t[k] = d[j++]; continue}
        if (j > right) {t[k] = d[i++]; continue}
        t[k] = (d[i] < d[j]) ? d[i++] : d[j++]
    }
    for (k = left; k <= right; k++) d[k] = t[k]
}

```

$$n/2 \leq \#cmps \leq n - 1$$



# Merge Sort : เวลาการทำงาน

```
mergeSort( d[1..n], left, right ) {  
    if (left >= right) return;  
    mid = ⌊(left + right) / 2⌋;  
    mergeSort(d, left, mid);  
    mergeSort(d, mid + 1, right);  
    merge(d, left, mid, right);  
}
```

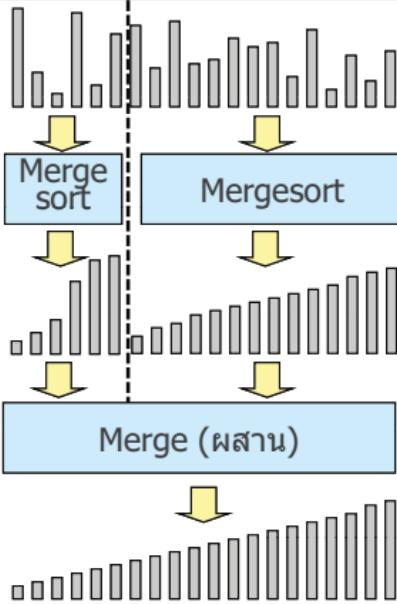
ให้  $t(n)$  คือ เวลา  
ในการ mergesort  
ข้อมูลจำนวน  $n$  ตัว

$$t(n) = 2t(n/2) + \Theta(n)$$

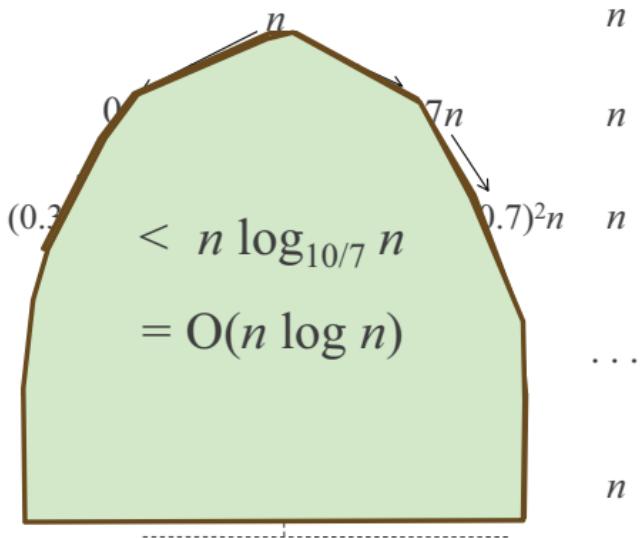
master method :  $n = \Theta(n^{\log_2 2})$

ได้  $t(n) = \Theta(n \log n)$

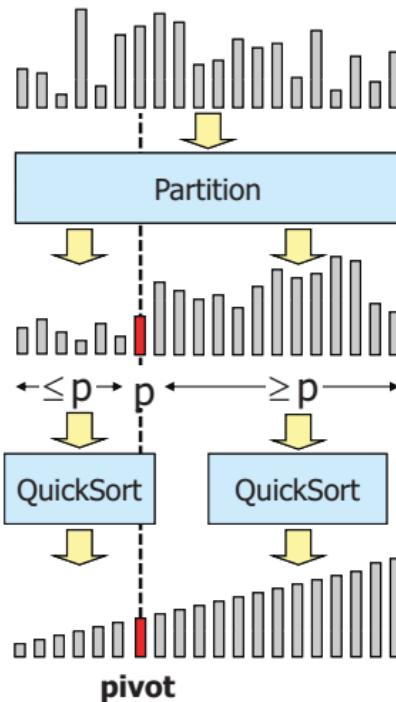
# Merge Sort : ถ้าแบ่ง 30 - 70



$$t(n) = t(0.3n) + t(0.7n) + \Theta(n)$$



# Quick Sort



# Quick Sort

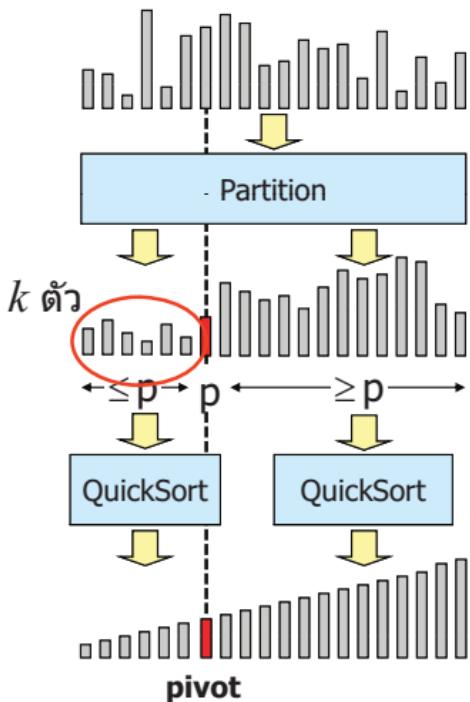
```
quickSort(d[1..n], left, right) {  
    if (left >= right) return  
    j = partition(d, left, right)  
    quickSort(d, left, j - 1)  
    quickSort(d, j + 1, right)  
}
```

# การแบ่งส่วน (partition)

d	12	5	15	18	0	9	11	52
---	----	---	----	----	---	---	----	----

```
partition( d[1..n], left, right ) {
    p = d[left]
    i = left, j = right + 1
    while (i < j) {
        while (d[--j] > p) ;
        while (d[++i] < p) if (i == right) break
        if (i < j) d[i] ↔ d[j]
    }
    d[left] ↔ d[j]
    return j
}
```

# Quick Sort : วิเคราะห์



ให้  $c(n)$  คือ #cmps ในการ quicksort ข้อมูลจำนวน  $n$  ตัว

$$n - 1$$

$$c(n) = c(k) + c(n-k-1) + n-1$$

$$c(k) + c(n - k - 1)$$

# Quick Sort : กรณีเร็วสุด

หลัง partition จำนวนข้อมูลของทั้งสองเท่ากัน

$$c(n) = c(k) + c(n - k - 1) + (n - 1)$$

$$\begin{aligned}c_{min}(n) &= c_{min}(n/2) + c_{min}(n - n/2 - 1) + (n - 1) \\&= 2c_{min}(n/2) + (n - 1) \\&= \Theta(n \log n)\end{aligned}$$

# Quick Sort : กรณีช้าสุด

หลัง partition จำนวนข้อมูลของฝั่งซ้ายหรือขวาเป็น 0

$$c(n) = c(k) + c(n - k - 1) + (n - 1)$$

$$c_{max}(n) = c(0) + c_{max}(n - 1) + (n - 1)$$

$$= c_{max}(n - 1) + (n - 1)$$

$$= c_{max}(n - 2) + (n - 2) + (n - 1)$$

...

$$= c_{max}(1) + 1 + 2 + \dots + (n - 2) + (n - 1)$$

$$= n(n - 1)/2$$

$$= \Theta(n^2)$$

# Quick Sort : กรณีเฉลี่ย

$$c(n) = c(k) + c(n - k - 1) + (n - 1)$$

$$c_{\text{avg}}(n) = \frac{1}{n} \sum_{k=0}^{n-1} (c_{\text{avg}}(k) + c_{\text{avg}}(n - k - 1)) + (n - 1)$$

$$= \frac{2}{n} \sum_{k=0}^{n-1} c_{\text{avg}}(k) + (n - 1)$$

$$nc_{\text{avg}}(n) = 2 \sum_{k=0}^{n-1} c_{\text{avg}}(k) + n(n - 1)$$

$$(n - 1)c_{\text{avg}}(n - 1) = 2 \sum_{k=0}^{n-2} c_{\text{avg}}(k) + (n - 1)(n - 2)$$

$$nc_{\text{avg}}(n) = (n + 1)c_{\text{avg}}(n - 1) + 2(n - 1)$$

# Quick Sort : กรณีเฉลี่ย

$$nc_{\text{avg}}(n) = (n+1)c_{\text{avg}}(n-1) + 2(n-1)$$

$$\begin{aligned} \frac{c_{\text{avg}}(n)}{n+1} &= \frac{c_{\text{avg}}(n-1)}{n} + \frac{2(n-1)}{n(n+1)} \\ &= \frac{c_{\text{avg}}(1)}{2} + 2 \sum_{i=2}^n \frac{(i-1)}{i(i+1)} \\ &\approx 2 \sum_{i=2}^n \frac{1}{(i+2)} \\ &= 2(\ln n + \Theta(1)) \end{aligned}$$

$$c_{\text{avg}}(n) = 2n \ln n + \Theta(n)$$

$$\approx 1.39n \log_2 n + \Theta(n)$$

$$= \Theta(n \log n)$$

# Quick Sort : เวลาการทำงาน

## ❖ Quicksort

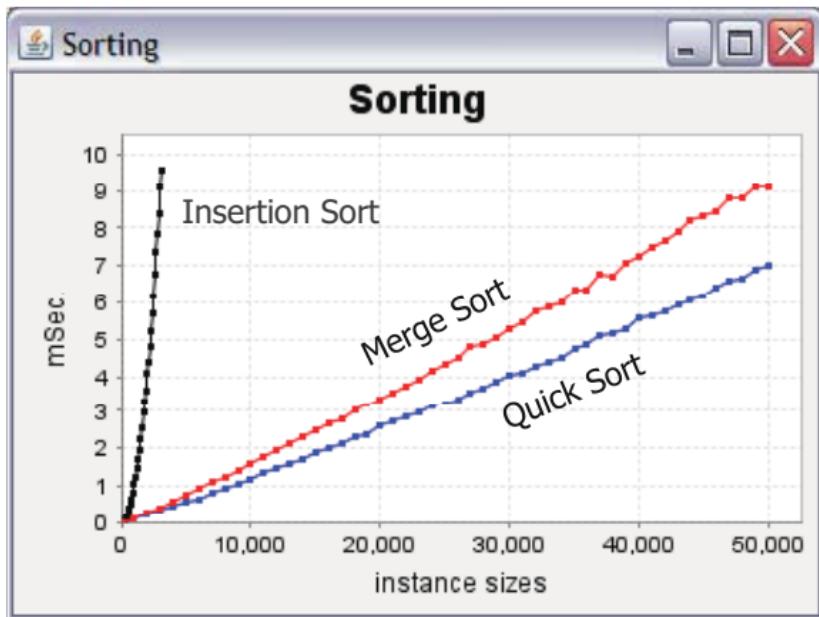
- ❖ กรณีช้าสุด :  $\Theta(n^2)$
- ❖ กรณีเร็วสุด :  $\Theta(n \log n)$
- ❖ กรณีเฉลี่ย :  $\Theta(n \log n)$

## ❖ เพื่อหลีกเลี่ยงกรณีช้าสุด เลือก pivot แบบสุ่ม

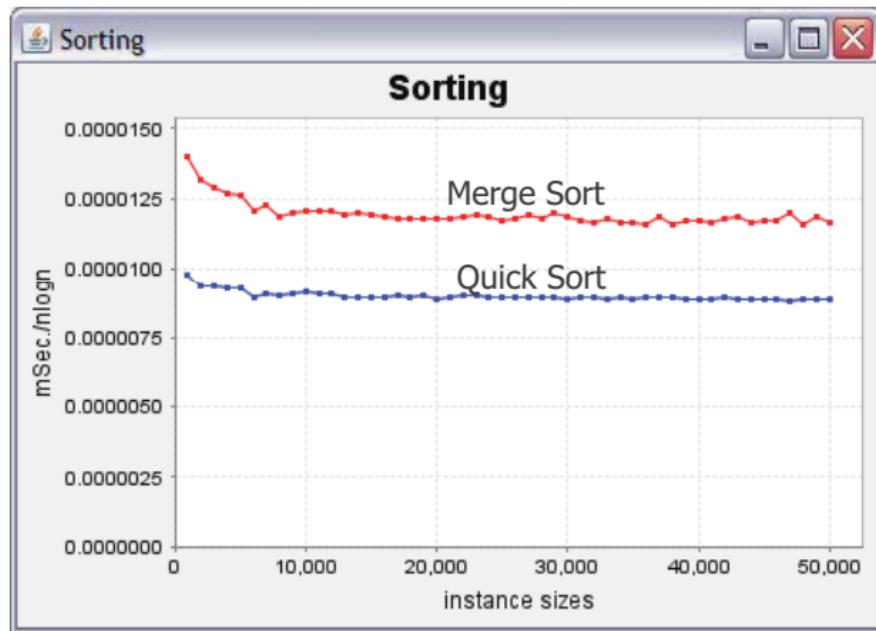
- ❖ มีโอกาสสูงที่ทำให้ quicksort ทำงานในเวลา  $\Theta(n \log n)$

```
partition( d[1..n] , left, right ) {  
    k = random(left, right)  
    d[k] ↔ d[left]  
    p = d[left]  
    i = left, j = right + 1  
    ...
```

# เวลาการทำงาน



# เวลา / $n \log n$

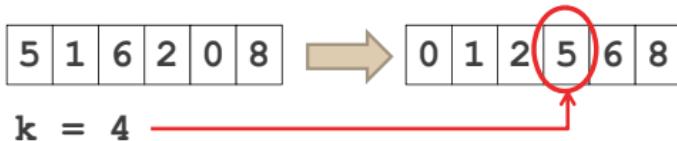


## Selection : หาตัวน้อยสุดอันดับ k

- ❖ **Input** :  $D = \langle d_1, d_2, \dots, d_n \rangle$  มีค่าต่างกันหมด  
จำนวนเต็ม  $k$ ,  $1 \leq k \leq n$
- ❖ **Output** : ข้อมูลใน  $D$  ที่มีค่าน้อยสุดอันดับที่  $k$   
(ข้อมูลใน  $D$  ที่มีมากกว่า  $d$  อื่นใน  $D$   
เป็นจำนวน  $k-1$  ตัว)
- ❖ **ตัวอย่าง** :  $D = \langle 4, 0, 2, 6, 8, 9, 7 \rangle$ ,  $k = 5$

ผลคือ 7 (0, 2, 4, 6 มีค่าน้อยกว่า 7  
นอกนั้นมากกว่า 7 หมด)

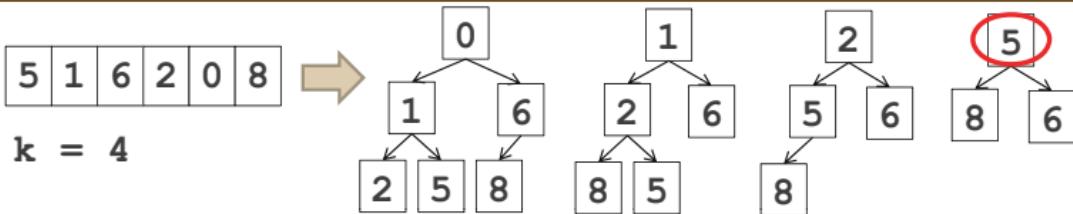
# sort ได้ d[k] คือคำตอบ



```
select( d[1..n], k ) {  
    mergeSort( d )  
    return d[k]  
}
```

$\Theta(n \log n)$

# ใช้ min-heap



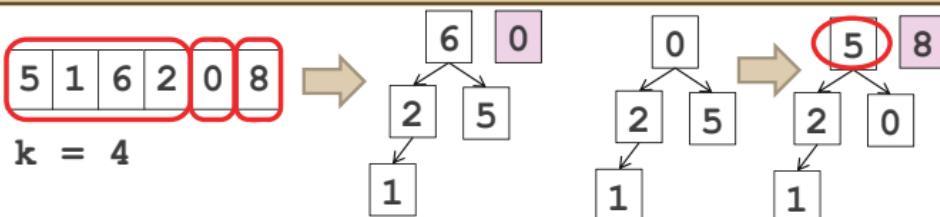
```

select( d[1..n], k ) {
    h = buildMinHeap( d )
    for (i = 1; i <= k-1; i++) {
        h.removeMin()
    }
    return h.getMin()
}

```

 $O(n + k \log n)$

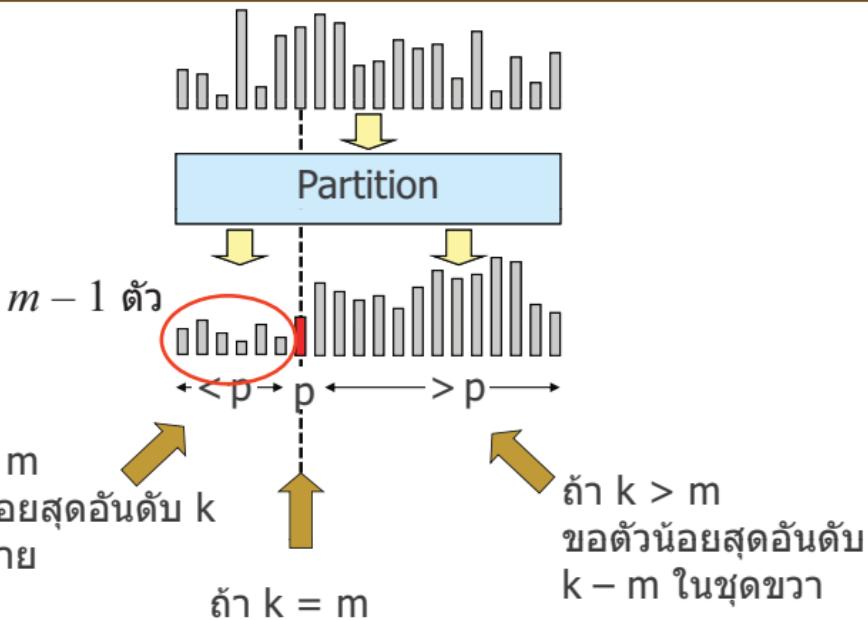
# ใช้ max-heap



```
select( d[1..n], k ) {
    h = buildMaxHeap( d[1..k] )
    for (i = k+1; i <= n; i++) {
        if (d[i] < h.getMax()) {
            h.removeMax()
            h.add( d[i] )
        }
    }
    return h.getMax()
}
```

$O(k + n \log k)$

# QuickSelect



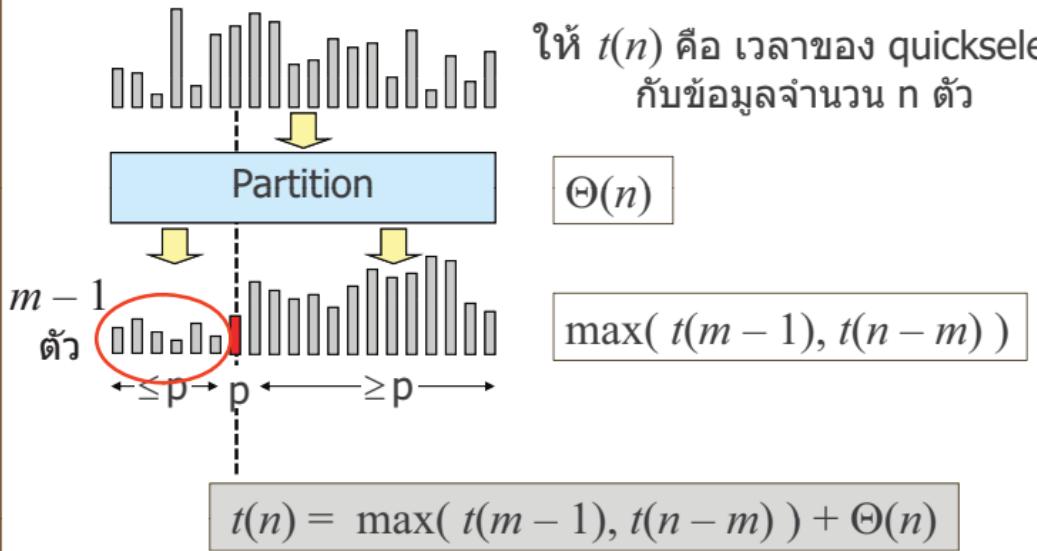
# QuickSelect

```
quickSelect( d[1..n] , left, right, k ) {  
    if (left == right) return d[left];  
    j = partition(d, left, right);  
    m = j - left + 1;  
    if (k == m) return d[j];  
    if (k < m)  
        return quickSelect(d, left, j-1, k);  
    else  
        return quickSelect(d, j+1, right, k-m);  
}
```

ตัวน้อยสุดอันดับ  $k$  จาก  $d[\text{left}]$  ถึง  $d[\text{right}]$

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

# QuickSelect : วิเคราะห์



## QuickSelect : กรณีเร็วสุด

pivot ที่หาได้ ไม่ใช่ตัวที่ต้องการ partition แล้วจำนวนข้อมูลของทั้งสองฝั่งเท่ากัน

$$t(n) \leq \max( t(m-1), t(n-m) ) + \Theta(n)$$

$$t_{min}(n) \leq t_{min}(n/2) + \Theta(n)$$

$$= O(n) \quad (\text{Master method})$$

# QuickSelect : กรณีซ้ำสุด

หลัง partition,  $m = 1$  หรือ  $n$

$$t(n) = \max( t(m-1), t(n-m) ) + \Theta(n)$$

$$t_{\max}(n) = t_{\max}(n-1) + \Theta(n)$$

$$= t_{\max}(n-2) + \Theta(n-1) + \Theta(n)$$

...

$$= t_{\max}(1) + \sum_{j=2}^n \Theta(j)$$

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

$$= \Theta(n^2)$$

# QuickSelect : กรณีเฉลี่ย

- ❖ เลือก pivot สุ่ม ๆ
- ❖ มีโอกาส  $1/2$  ที่ได้ pivot เป็นอันดับ **25% - 75%**
  - ❖ ถ้า pivot เป็น
    - ❖ ตัวที่อันดับ 25% ฝั่งขวา มี 75% ของจำนวนข้อมูลทั้งหมด
    - ❖ ตัวที่อันดับ 75% ฝั่งซ้าย มี 75% ของจำนวนข้อมูลทั้งหมด
- ❖ จะได้  $t(n) \leq t(3n/4) + \Theta(n)$ ,  $c = \log_{4/3} 1 = 0$   
 $n = \Omega(n^{0+\epsilon})$ ,  $(3n/4 - 1) \leq (n - 1)$ ,  $n \geq 0$ ,  $\rightarrow O(n)$
- ❖ โยนเหรียญ 1 ครั้ง มีโอกาส  $1/2$  ได้น้ำ อย่างใดน้ำ ต้องโยนเหรียญโดยเฉลี่ย 2 ครั้ง ( $E = 1 + E/2$ )
- ❖ โดยเฉลี่ยการแบ่งส่วนทุก 2 ครั้ง ได้การแบ่งที่ "ดี"

ช่วงกว้าง 50% ของข้อมูล

เป็นการแบ่งที่ "ดี"

# QuickSelect : กรณีซ้ำสุด $O(n)$

- ❖ หาวิธี partition ที่แบ่งได้ "ดี" แน่ ๆ
- ❖ ใช้ "mm5" เป็น pivot ในการแบ่ง
  - ❖ ประกันว่าจะแบ่ง 30% – 70% แม้กรณีแย่สุด
  - ❖  $t(n) = t(0.7n) + \text{เวลาในการหา "mm5"} + \Theta(n)$
  - ❖ จะเสนอวิธีหา mm5 ที่ใช้เวลา  $t(0.2n)$
  - ❖ จะได้  $t(n) = t(0.7n) + t(0.2n) + \Theta(n)$   
 $= \Theta(n)$

Median-of-Median-of-Five

## Median-of-Median-of-Five

- แบ่งข้อมูลทั้งหมดเป็นชุด ๆ ชุดละ 5 ตัว
- หาน้อยสูนของข้อมูลในแต่ละชุด
- mm5 คือมัธยฐานที่ห่างจากข้อมูลที่เป็นมัธยฐานของแต่ละชุด

10, 11, 3, 2, 5 6, 19, 20, 13 8 7, 1 9, 14, 15,

4, 31, 34, 25, 28 21, 43, 12, 16, 18 17, 33, 90, 81, 0,

61, 72, 37

5, 13, 9, 28, 18 33, 61

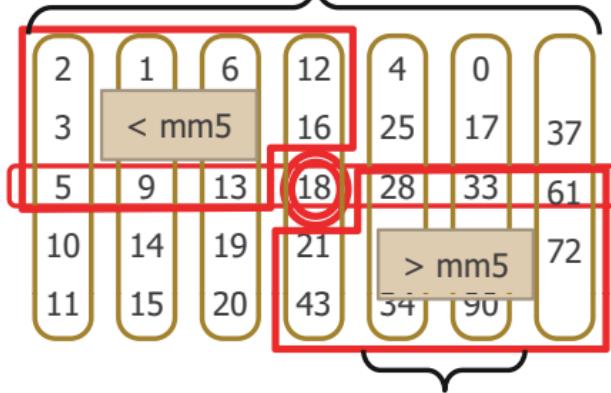
mm5

# Median-of-Median-of-Five

- ❖ มีຂໍ້ມູລຍ່າງນ້ອຍ 30% ທີ່ມີຄ່ານ້ອຍກວ່າ mm5
- ❖ มีຂໍ້ມູລຍ່າງນ້ອຍ 30% ທີ່ມີຄ່າມາກກວ່າ mm5

ມີຂໍ້ມູລ  $n$  ຕົວ

ມີ  $\lceil n/5 \rceil$  ກລຸມ



ມີ  $\lceil n/10 \rceil - 2$  ກລຸມ ມີ  $3\lceil n/10 \rceil - 6$  ຕົວ

# เวลาในการหา mm5

## 1. แบ่งข้อมูลทั้งหมดเป็นชุด ๆ ชุดละ 5 ตัว

- ❖ ได้ทั้งหมด  $\lceil n/5 \rceil$  ชุด

$\Theta(n)$

## 2. หามัธยฐานของข้อมูลในแต่ละชุด

- ❖ หามัธยฐานของข้อมูล 5 ตัวใช้  $\Theta(1)$
- ❖ หาของ  $\lceil n/5 \rceil$  ชุด
- ❖ ได้มัธยฐานจำนวน  $\lceil n/5 \rceil$  ตัว

$\Theta(n)$

## 3. หา mm5 จากข้อมูลที่เป็นมัธยฐานของแต่ละชุด

- ❖ ใช้ quickselect หา median ของข้อมูลจำนวน  $\lceil n/5 \rceil$  ตัว ที่ได้ในข้อ 2)
- ❖ ให้  $t(n)$  คือเวลาเพื่อ quickSelect กับข้อมูล  $n$  ตัว

$t(n/5)$

$t(n/5) + \Theta(n)$

# QuickSelect



- ❖ หา mm5 :  $t(n/5) + \Theta(n)$
- ❖ ใช้ mm5 partition :  $\Theta(n)$
- ❖ quickSelect ในชุดใหญ่ต่อ (อย่างมาก 70%)  
:  $t(0.7n)$
- ❖ เวลารวมของ quickSelect กรณีเข้าสุด

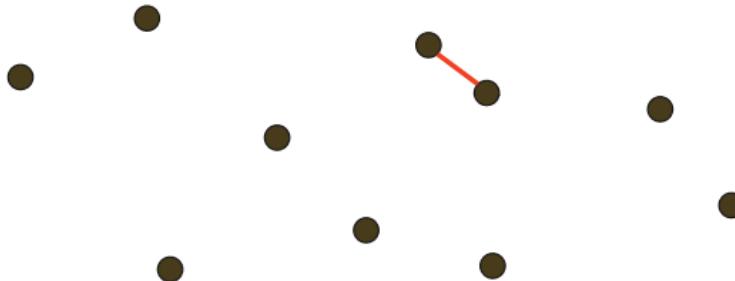
$$\begin{aligned}
 t(n) &= t(0.7n) + t(0.2n) + \Theta(n) \\
 &= \Theta(n)
 \end{aligned}$$

ได้ผลเช่นไร ถ้าใช้ mm7, mm9, mm3 ?  
น่าใช้ mm5 กับ quicksort ?

# Closest Point

- ❖ **Input** : จุด  $n$  จุดบนระนาบสองมิติ
- ❖ **Output** : ระยะทางระหว่างคู่จุดที่ใกล้กันที่สุด

$$d_{i,j} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$



# จุดที่ใกล้ที่สุด

- ❖ คำนวณระยะทางของคู่จุดทุกคู่
- ❖ ต้องลอง  $n(n-1)/2$  คู่ ใช้เวลา  $\Theta(n^2)$

```

closestPair( x[1..n], y[1..n] ) {
    min = ∞
    for (i = 1; i <= n; i++)
        for (j = i+1; j <= n; j++)
            dx = |x[i] - x[j]|
            dy = |y[i] - y[j]|
            d = sqrt(dx2 + dy2)
            if (d < min) min = d
    }
}
return min
}

```

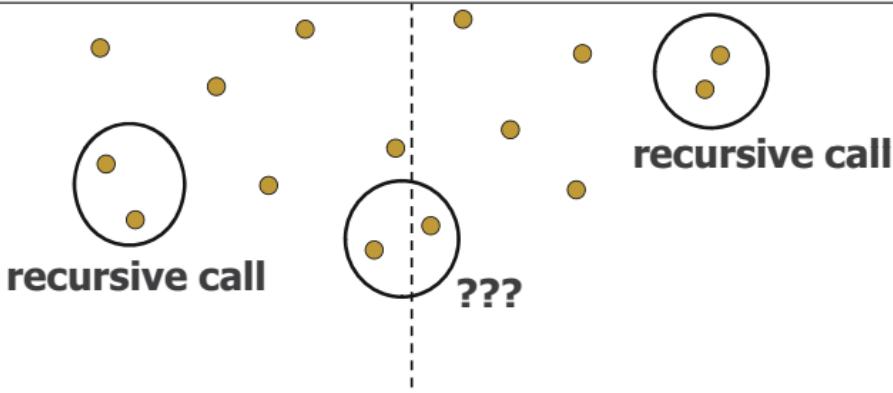
# Divide & Conquer

$$t(n) = 2t(n/2) + \Theta(?)$$

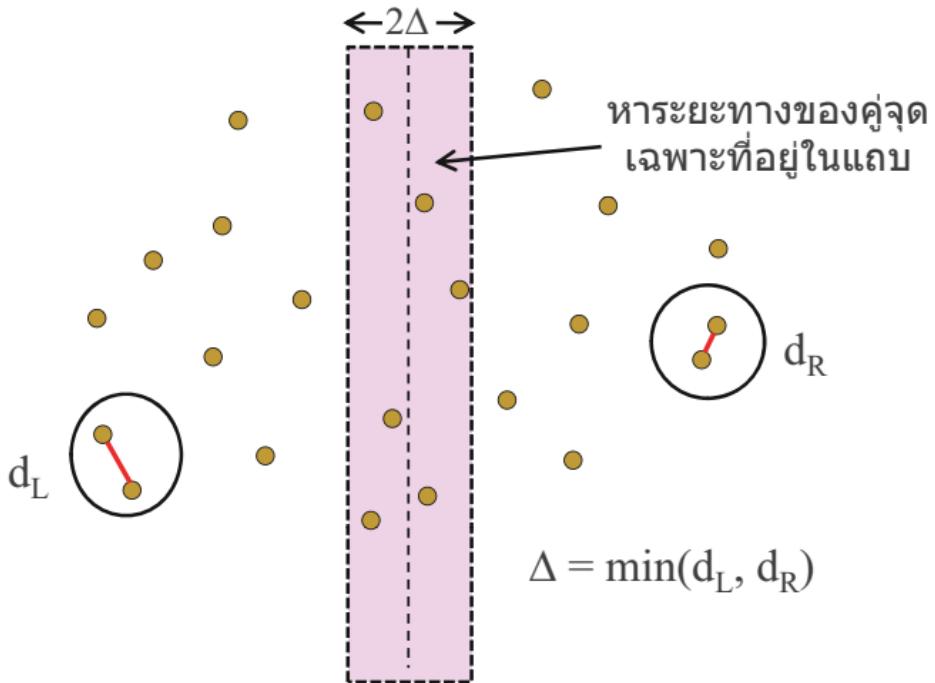
$$t(n) = 2t(n/2) + \Theta(n^2) = \Theta(n^2)$$

$$t(n) = 2t(n/2) + \Theta(n \log n) = \Theta(n \log^2 n)$$

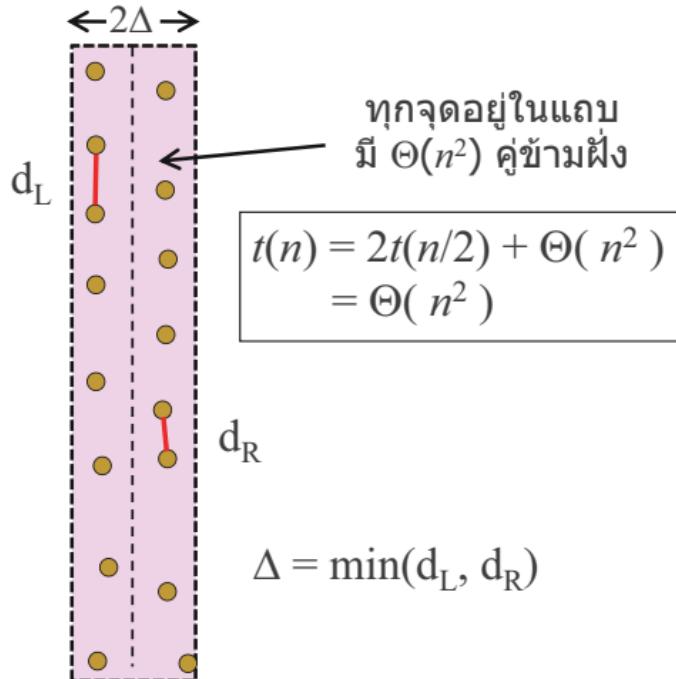
$$t(n) = 2t(n/2) + \Theta(n) = \Theta(n \log n)$$



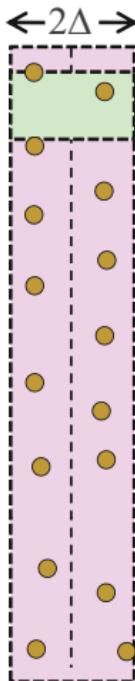
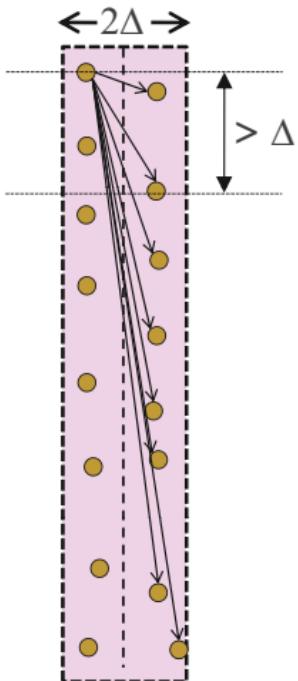
# Divide & Conquer



# คำนวณทุกคู่ที่ข้ามฝั่ง



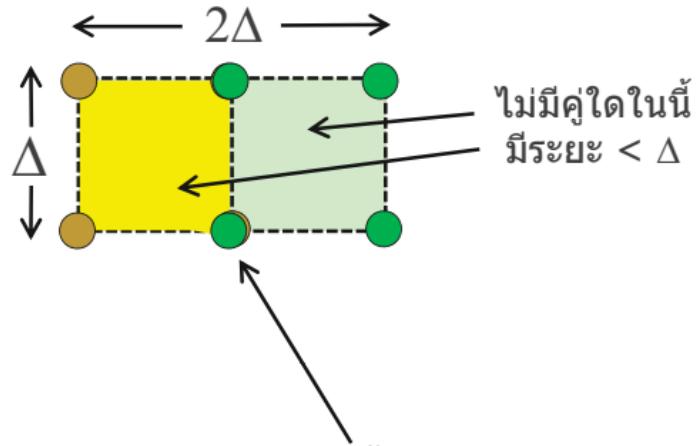
# ไม่จำเป็นต้องคำนวณทุกคู่ที่ข้ามฝั่ง



- มีจุดอยู่ในกรอบ  $2\Delta \times \Delta$  อย่างมาก 8 จุด
- คำนวณคู่จุดข้ามฝั่งในหนึ่งกรอบใช้เวลา  $\Theta(1)$
- ต้องเลื่อนกรอบ  $\Theta(n)$  ครั้ง
- มีจำนวน  $\Theta(n)$  คู่ที่ต้องคำนวณ

$$\begin{aligned} t(n) &= 2t(n/2) + \Theta(n) \\ &= \Theta(n \log n) \end{aligned}$$

## มีอย่างมาก 8 จุดในกรอบ $2\Delta \times \Delta$



ตำแหน่งนี้มีสองจุด  
โดยแต่ละจุดถูกแยก  
ไปอยู่คนละฝั่ง

$$t(n) = 2t(n/2) + \Theta(n) ?$$

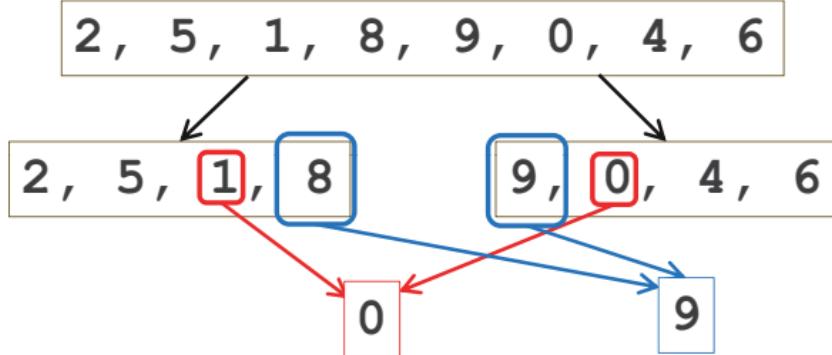
- ❖ เลื่อนกรอบจากบนลงล่างได้อย่างไร ?
- ❖ รู้ได้อย่างไรว่าจุดโดยยุ่งเหยิง จุดโดยยุ่งขวา
- ❖ ต้อง **sort** จุดตามแนวดิ่ง และแนวนอน
- ❖ ถ้า **sort** ทุกครั้งในทุกระดับของ **recursive**
  - ❖  $t(n) = 2t(n/2) + \Theta(n \log n) \rightarrow t(n) = \Theta(n \log^2 n)$
- ❖ **sort** ครั้งเดียวเก็บไว้ก่อนก็ได้
  - ❖ ทุกระดับของ recursive ต้องเทียบกับรายการที่ **sort** ไว้
  - ❖  $t(n) = \max( \Theta(n \log n), 2t(n/2) + \Theta(n) )$
  - ❖ เนื่องจาก  $t(n) = 2t(n/2) + \Theta(n) = \Theta(n \log n)$
  - ❖ ดังนั้น  $t(n) = \Theta(n \log n)$

# min + max

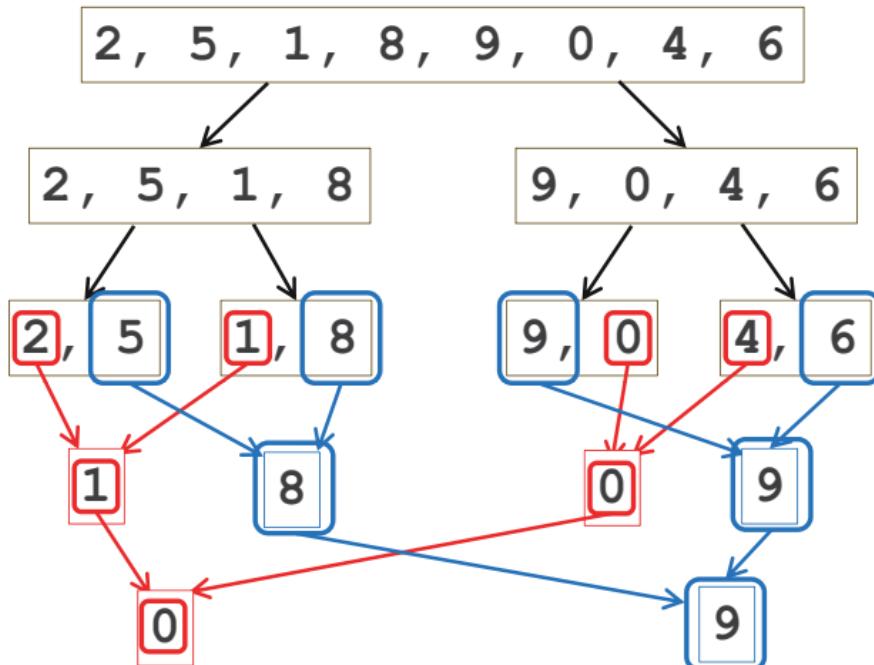
```
minmax( d[1..n] ) {  
    max = d[1], min = d[1]  
    for (k=2; k<=n; k++) {  
        if (d[k] > max) max = d[k]  
        if (d[k] < min) min = d[k]  
    }  
    return (min,max)  
}
```

$$\#cmps = 2(n - 1)$$

# min + max : ແບ່ງຄຽງ



## min + max : แบ่งครึ่ง ๆ



# min + max : แบ่งครึ่ง ๆ

```

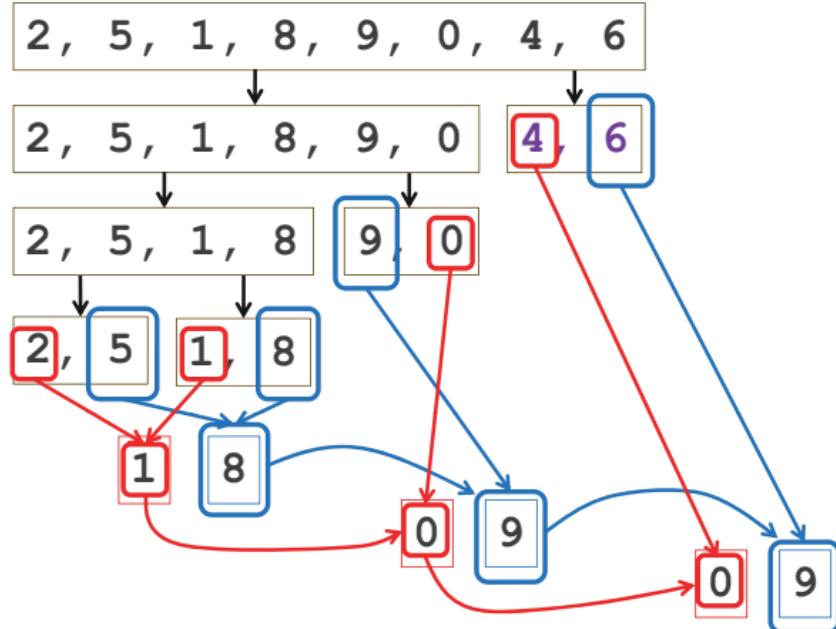
minmax1( d[1..n], left, right ) {
    if (right-left == 0) // มี 1 ตัว
        return (d[right], d[right])
    if (right-left == 1) { // มี 2 ตัว
        if (d[left] < d[right])
            return (d[left], d[right]);
        else
            return (d[right], d[left]);
    }
    mid = ⌊(left +right)/2⌋
    (min1, max1) = minmax1(d, left, mid)
    (min2, max2) = minmax1(d, mid + 1, right)
    return (min(min1, min2), max(max1, max2));
}

```

$$c(n) = c(\lceil n/2 \rceil) + c(\lfloor n/2 \rfloor) + 2$$

$$c(1) = 0, c(2) = 1$$

# $\min + \max : \text{แบ่ง } (n-2) + 2$



# min + max : แบบ $(n-2) + 2$

```

minmax2( d[1..n], right ) {
    if (right == 1)
        return (d[right], d[right])
    if (d[right-1] < d[right])
        min1 = d[right-1], max1 = d[right];
    else
        min1 = d[right], max1 = d[right-1];
    if (right > 2) {
        (min2, max2) = minmax2(d, right - 2);
        min1 = min(min1, min2);
        max1 = max(max1, max2);
    }
    return (min1, max1);
}

```

$$c(n) = c(n-2) + 3$$

$$c(1) = 0, c(2) = 1$$

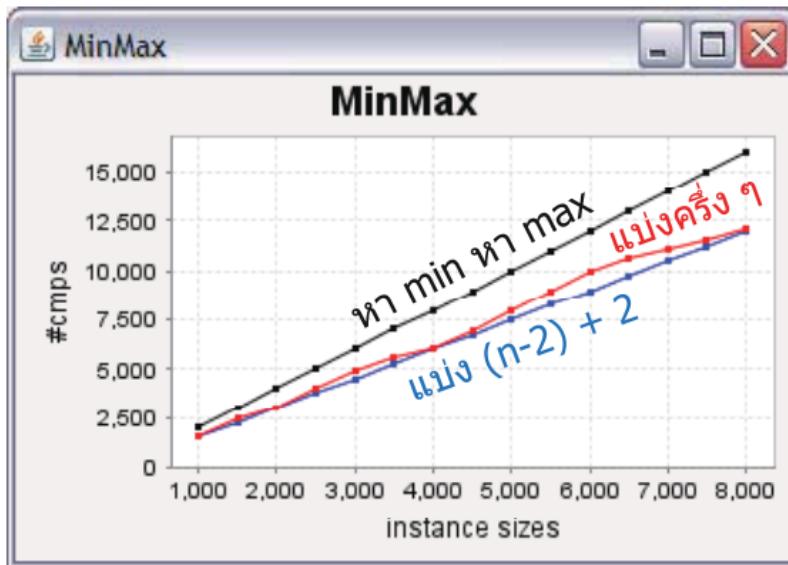
# วิเคราะห์: แบ่ง $(n-2)$ + 2

$$c(n) = c(n - 2) + 3, \quad c(1) = 0, \quad c(2) = 1$$
$$c(n) = \lceil 3n / 2 \rceil - 2$$

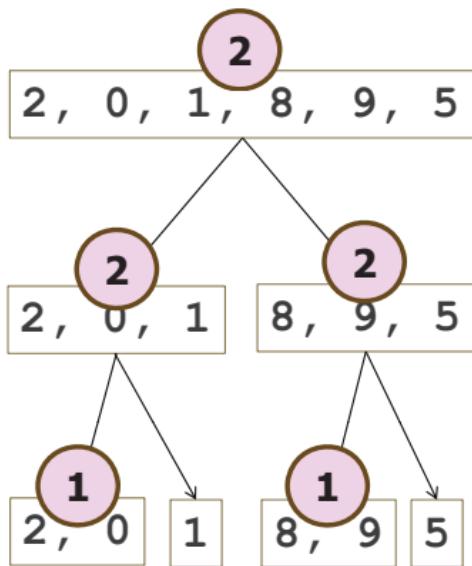
$n$	$c(n)$
1	0
3	3
5	6
7	9
9	12
...	...

$n$	$c(n)$
2	1
4	4
6	7
8	10
10	13
...	...

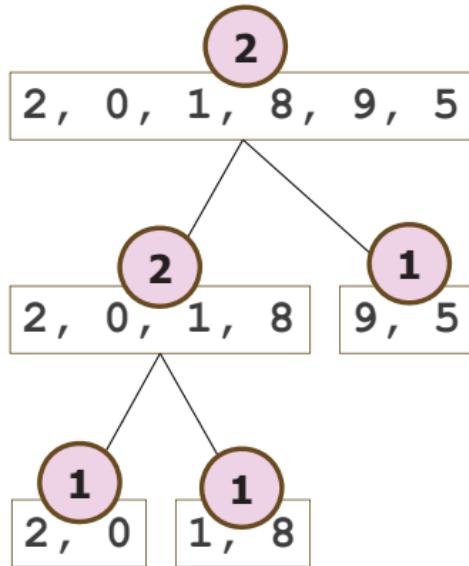
# เปรียบเทียบ



# แบ่งครึ่งเหลือเศษมากกว่า



เปรียบเทียบ 8 ครั้ง



เปรียบเทียบ 7 ครั้ง

# Celebrity Problem

- ❖ อยากรู้ว่าใครคือ "ดาวเด่น" ของงานสังคมที่มีผู้ร่วมงาน  $n$  คน
- ❖ "ดาวเด่น" คือผู้ซึ่งทุก ๆ คนในงานรู้จักแต่เจ้าตัวไม่รู้จักใครเลย

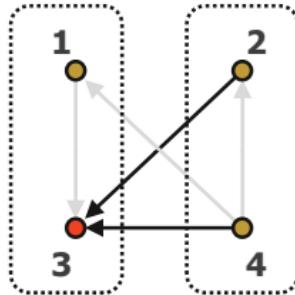
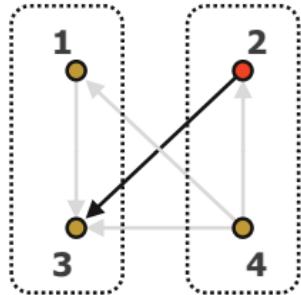
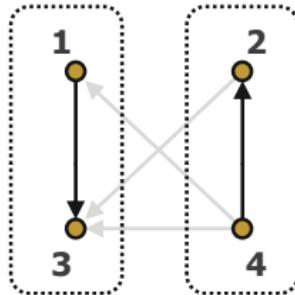
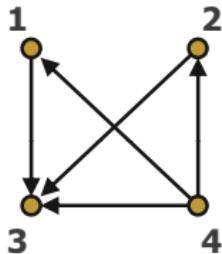


# คุย

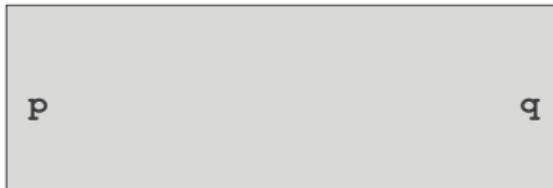
```
Celebrity ( a[1..n, 1..n] ) {  
    for i = 1 to n {  
        c1 = 0  
        for k = 1 to n  
            c1 = c1 + a[k,i]  
        c2 = 0  
        for k = 1 to n  
            c2 = c2 + a[i,k]  
        if (c1 == n-1 AND c2 == 0) return i  
    }  
    return 0  
}
```

	1	2	3	4
1	0	0	1	0
2	0	0	1	0
3	0	0	0	0
4	1	1	1	0

# Divide & Conquer

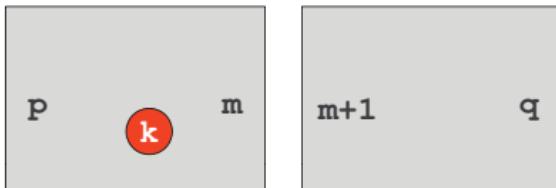


## Divide & Conquer (50 : 50)



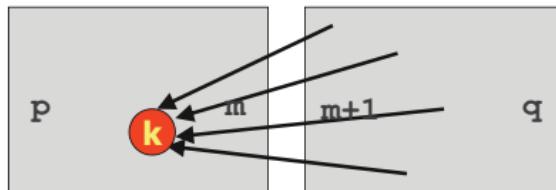
```
Celebrity_DQ( a[1..n, 1..n], p, q ) {
```

## Divide & Conquer (50 : 50)



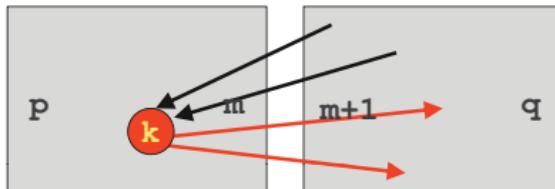
```
Celebrity_DQ( a[1..n, 1..n], p, q ) {  
    if ( p == q ) return p  
    m = (p+q)/2  
    k = Celebrity_DQ( a, p, m )
```

# Divide & Conquer (50 : 50)



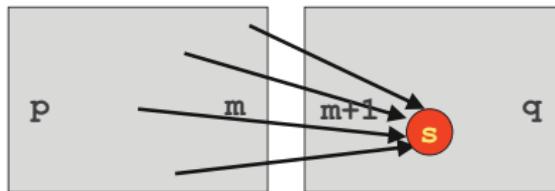
```
Celebrity_DQ( a[1..n], 1..n ], p, q ) {  
    if ( p == q ) return p  
    m = (p+q)/2  
    k = Celebrity_DQ( a, p, m )  
    k = Celebrity_Check( a, k, m+1, q )  
    if ( k > 0 ) return k
```

# Divide & Conquer (50 : 50)



```
Celebrity_DQ( a[1..n], 1..n ], p, q ) {  
    if ( p == q ) return p  
    m = (p+q)/2  
    k = Celebrity_DQ( a, p, m )  
    k = Celebrity_Check( a, k, m+1, q )  
    if ( k > 0 ) return k
```

# Divide & Conquer (50 : 50)



```
Celebrity_DQ( a[1..n], 1..n ], p, q ) {  
    if ( p == q ) return p  
    m = (p+q)/2  
    k = Celebrity_DQ( a, p, m )  
    k = Celebrity_Check( a, k, m+1, q )  
    if ( k > 0 ) return k  
    s = Celebrity_DQ( a, m+1, q )  
    s = Celebrity_Check( a, s, p, m )  
    return s  
}
```

# Divide & Conquer (50 : 50)



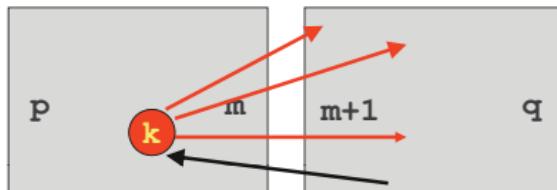
```
Celebrity_DQ( a[1..n, 1..n], p, q ) {
    if ( p == q ) return p
    m = (p+q)/2
    k = Celebrity_DQ( a, p, m )
    k = Celebrity_Check( a, k, m+1, q )
    if ( k > 0 ) return k
    s = Celebrity_DQ( a, m+1, q )
    s = Celebrity_Check( a, s, p, m )
    return s
}
```

# Divide & Conquer (50 : 50)



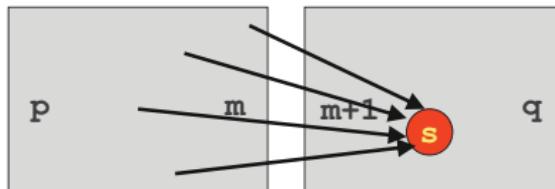
0	0	1	0	0	1	1	1
1	0	1	1	1	0	1	0
0	0	0	0	0	1	1	1
1	0	1	0	1	1	1	0
1	1	0	0	0	1	1	1
1	0	0	0	0	0	1	1
0	0	0	0	0	0	0	0
1	0	1	0	0	1	1	0

# Divide & Conquer (50 : 50)



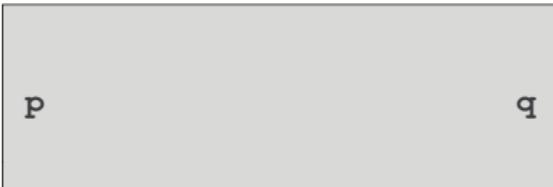
0	0	1	0	0	1	1	1
1	0	1	1	1	0	1	0
0	0	0	0	0	1	1	1
1	0	1	0	1	1	1	0
1	1	0	0	0	1	1	1
1	0	0	0	0	0	1	1
0	0	0	0	0	0	0	0
1	0	1	0	0	1	1	0

## Divide & Conquer (50 : 50)



0	0	1	0	0	1	1	1	1
1	0	1	1	1	0	1	0	0
0	0	0	0	0	1	1	1	1
1	0	1	0	1	1	1	0	0
1	1	0	0	0	1	1	1	1
1	0	0	0	0	0	0	1	1
0	0	0	0	0	0	0	0	0
1	0	1	0	0	1	1	0	0

## Divide & Conquer (1 : n - 1)



```
Celebrity_DQ( a[1..n, 1..n], p, q ) {
```

# Divide & Conquer (1 : n - 1)



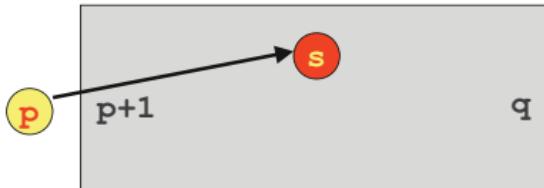
```
Celebrity_DQ( a[1..n], 1..n ], p, q ) {  
    if ( p == q ) return p  
    k = Celebrity_Check( a, p, p+1, q )  
    if ( k > 0 ) return k
```

# Divide & Conquer (1 : n - 1)



```
Celebrity_DQ( a[1..n], 1..n ], p, q ) {  
    if ( p == q ) return p  
    k = Celebrity_Check( a, p, p+1, q )  
    if ( k > 0 ) return k  
    s = Celebrity_DQ( a, p+1, q )  
  
}
```

# Divide & Conquer (1 : n - 1)



```
Celebrity_DQ( a[1..n], 1..n ], p, q ) {  
    if ( p == q ) return p  
    k = Celebrity_Check( a, p, p+1, q )  
    if ( k > 0 ) return k  
    s = Celebrity_DQ( a, p+1, q )  
    s = Celebrity_Check( a, s, p, p )  
    return s  
}
```

## Divide & Conquer (1 : n - 1)

```
Celebrity_DQ( a[1..n, 1..n], p, q ) {  
    if ( p == q ) return p  
    k = Celebrity_Check( a, p, p+1, q )  
    if ( k > 0 ) return k  
    s = Celebrity_DQ( a, p+1, q )  
    s = Celebrity_Check( a, s, p, p )  
    return s  
}
```

# Decrease & Conquer

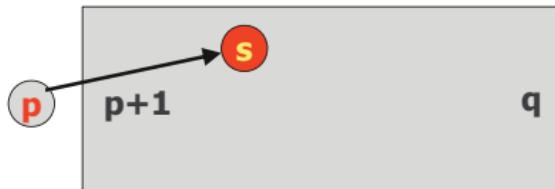
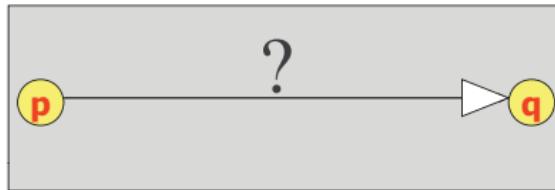
$$t(n) =$$

$$t(n) =$$

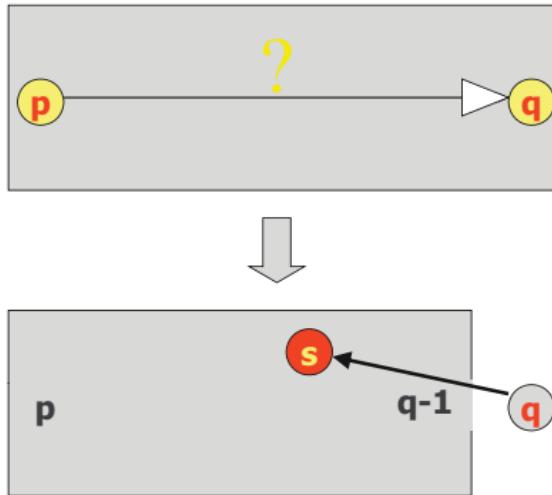
```
Celebrity_DQ( a[1..n], 1..n ], p, q ) {
    if ( p == q ) return p
    k = Celebrity_Check( a, p, p+1, q )
    if ( k > 0 ) return k
    s = Celebrity_DQ( a, p+1, q )
    s = Celebrity_Check( a, s, p, p )
    return s
}
```

จะเลือกตัวใด ให้แน่ใจว่าไม่ใช่ "ดาวเด่น" แน่ ๆ

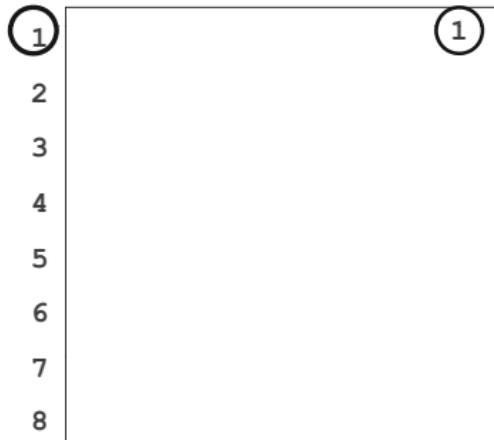
# Decrease & Conquer



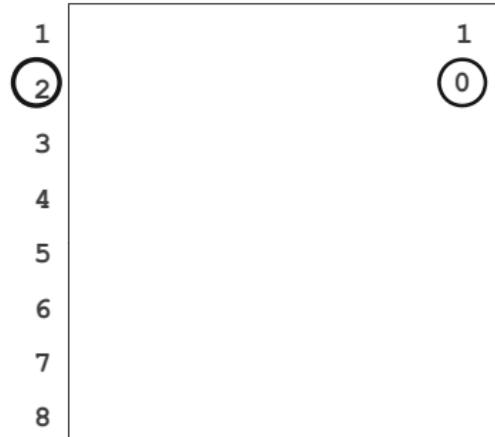
# Decrease & Conquer



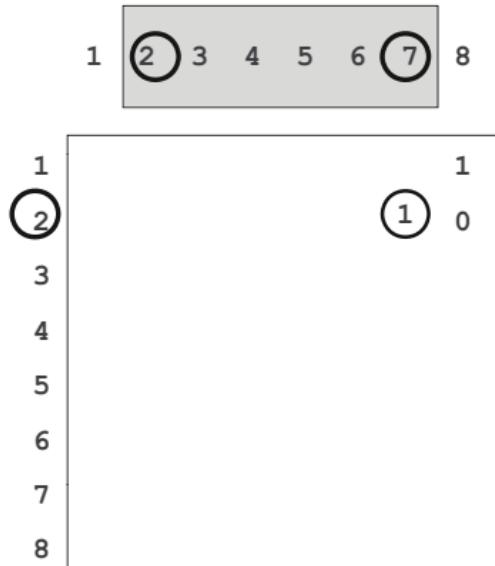
# Decrease & Conquer



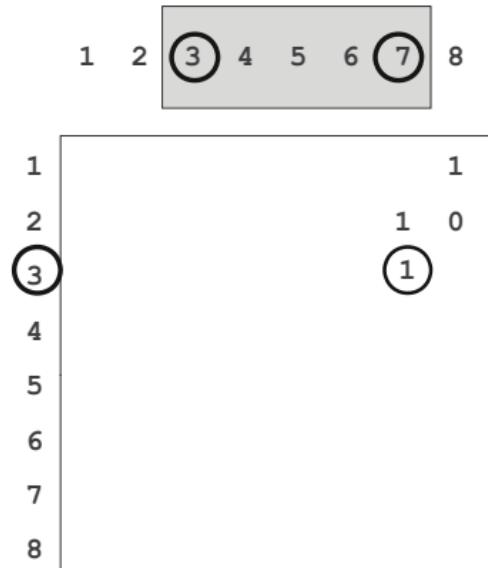
# Decrease & Conquer



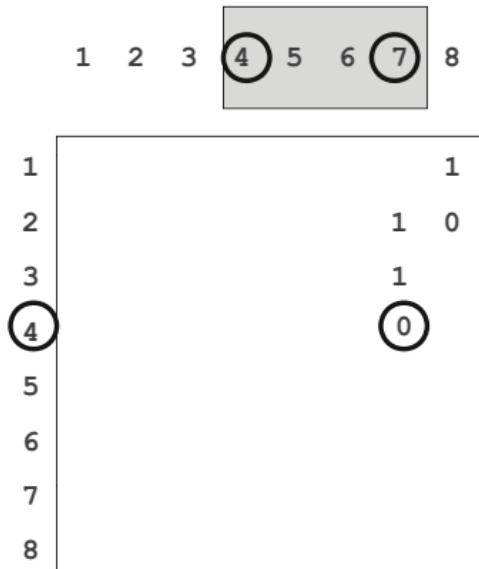
# Decrease & Conquer



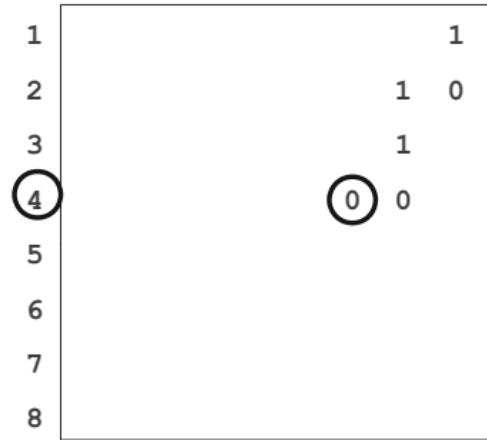
# Decrease & Conquer



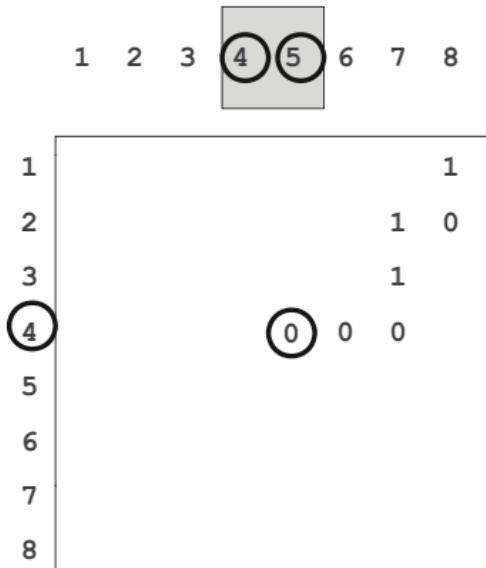
# Decrease & Conquer



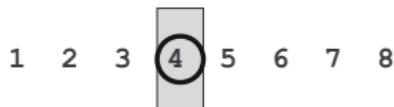
# Decrease & Conquer



# Decrease & Conquer

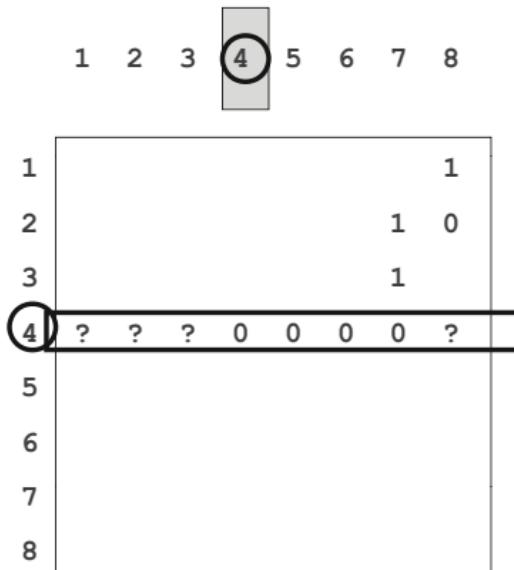


# Decrease & Conquer

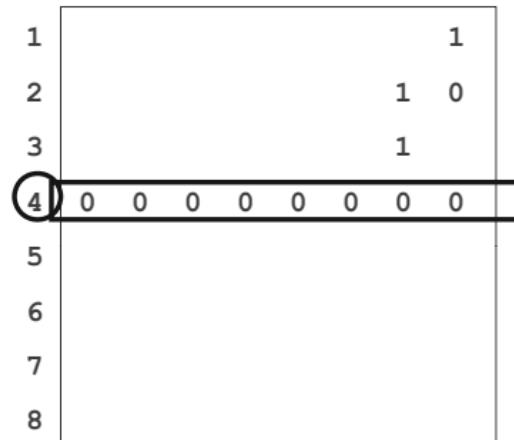
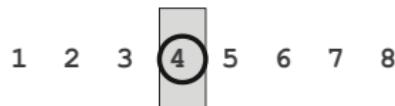


1			1
2			1 0
3			1
4		0 0	0
5			
6			
7			
8			

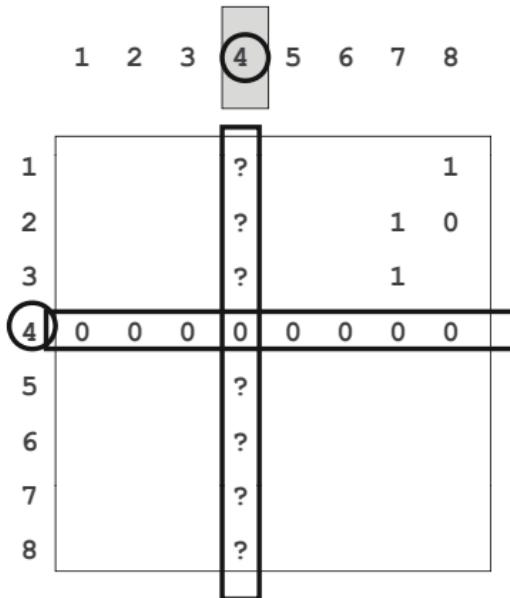
# Decrease & Conquer



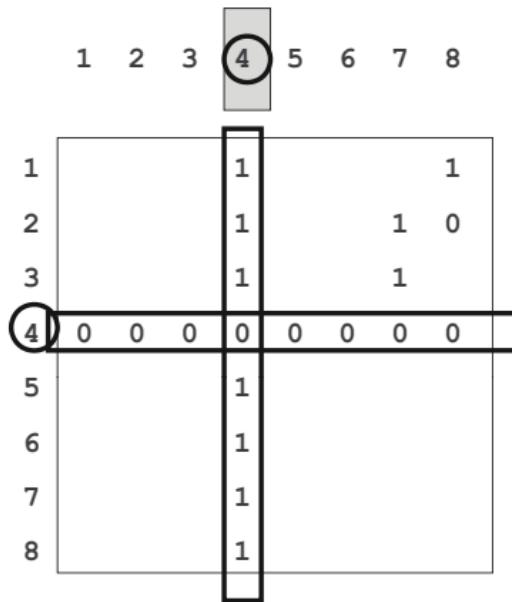
# Decrease & Conquer



# Decrease & Conquer



# Decrease & Conquer



# Decrease & Conquer

```
Celebrity_DQ( a[1..n], 1..n ] , p, q ) {  
  
    if ( p == q ) return p  
  
    if ( a[p,q] == 1) {  
        s = Celebrity_DQ( a, p+1, q )  
        s = Celebrity_Check( a, s, p, p )  
    } else {  
        s = Celebrity_DQ( a, p, q-1 )  
        s = Celebrity_Check( a, s, q, q )  
    }  
  
    return s  
}
```



จำนวน probes =  $p(n) = p(n-1)+3 = 3(n-1)$

# สรุป

- ❖ Divide + Conquer + Combine
- ❖ วิเคราะห์ด้วย Master method, recursion tree,
- ❖ หาว่าการะการแบ่งปัญหา การรวมค่าตอบ หรือการ แก้ปัญหาย่อย ส่วนใดเป็นตัวกำหนดประสิทธิภาพ
- ❖ โดยทั่วไปมักแบ่งปัญหาให้ญี่เป็นสองปัญหาย่อย แต่ควรลองแบบอื่น อาจดีกว่า

กำหนดการพlovat

**Dynamic Programming**

สมชาย ประสิทธิ์จูตระกูต

## หัวข้อ

- ❖ แบ่งให้ญี่เป็นย่อย ถ้ามีปัญหาย่อยที่เข้า จะเข้า
- ❖ จำคำตอบไว้ใช้ในอนาคต → เร็ว
- ❖ ใช้กับการแก้ปัญหาได้ทั้งแบบ
  - ❖ top-down
  - ❖ bottom-up
- ❖ กำหนดการพลวัต (**Dynamic Programming**)
- ❖ ตัวอย่าง

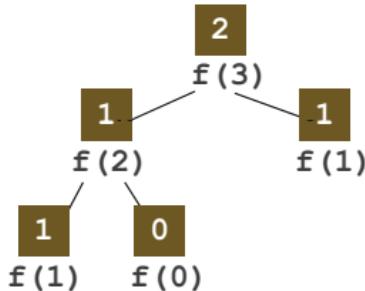
# จำนวน斐波นัคชี (Fibonacci)

$$f_n = f_{n-1} + f_{n-2}$$

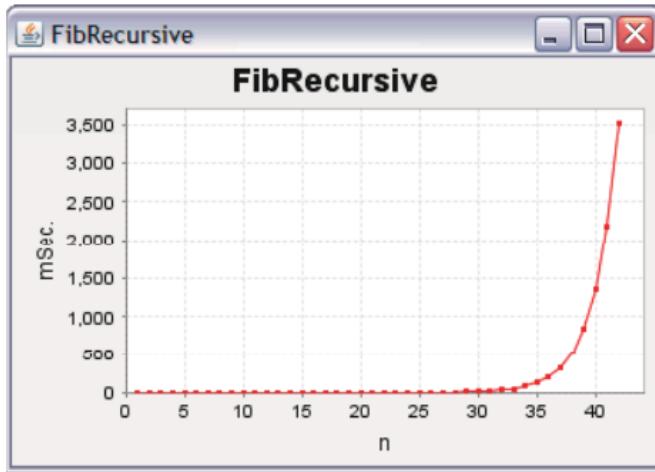
$$f_0 = 0, f_1 = 1$$

Top-down

```
f( n ) {
    if (n < 2) return n
    return f(n - 1) + f(n - 2)
}
```



# เวลาการทำงาน



Intel P8400 2.26GHz, Java 6u14

# วิเคราะห์เวลาการทำงาน

```
f( n ) {
    if (n < 2) return n
    return f(n - 1) + f(n - 2)
}
```

ให้  $t(n)$  แทนจำนวนครั้งที่เรียก  $f(n)$  จะได้

$$t(n) = t(n - 1) + t(n - 2) + 1, \quad t(0) = 1, \quad t(1) = 1$$

n =	0	1	2	3	4	5	6	7	8
$f(n) =$	0	1	1	2	3	5	8	13	21
$t(n) =$	1	1	3	5	9	15	25	41	67

$$t(n) = 2f(n+1) - 1$$

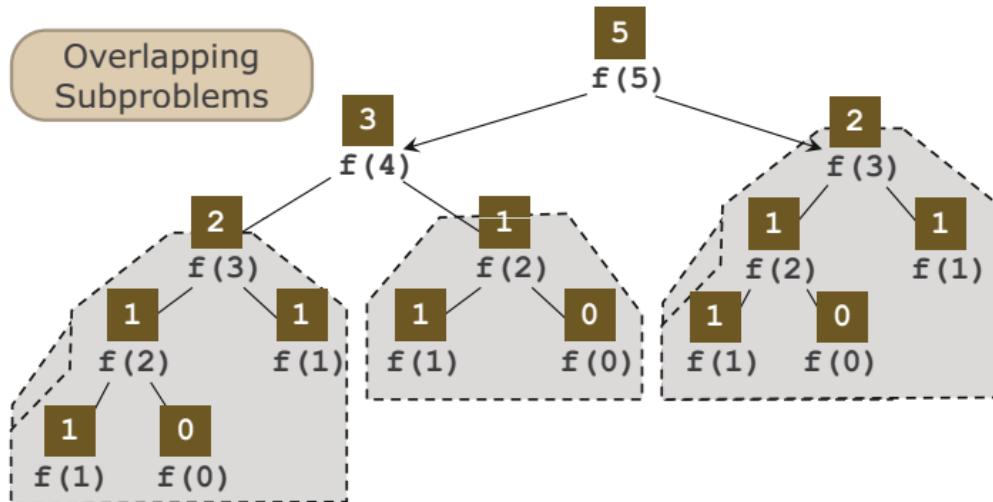
$$\Theta(\phi^n)$$

$$f(n) = \frac{\phi^n - (1-\phi)^n}{\sqrt{5}}, \quad \phi = \frac{1+\sqrt{5}}{2} \approx 1.61803\dots$$

# ຊ້າເພຣະຄໍານວณຊ້າ ۱

```
f( n ) {  
    if (n < 2) return n  
    return f(n - 1) + f(n - 2)  
}
```

Overlapping Subproblems

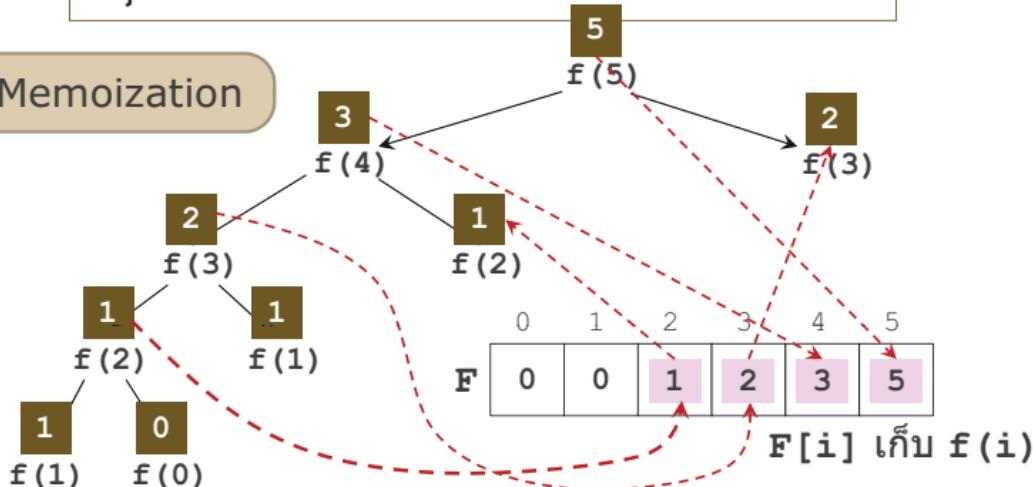


# หลักเลี้ยงการคำนวณซ้ำด้วยการจำ

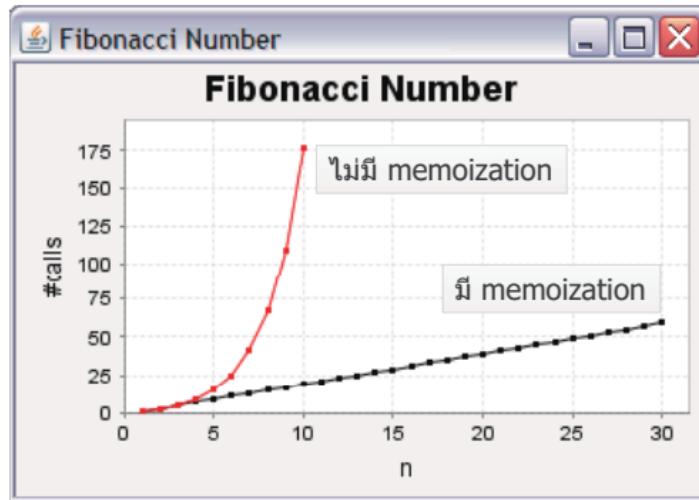
```
f(n, F[0..n] ) {
    if (n < 2) return n
    if (F[n] > 0) return F[n]
    return F[n] = f(n-1, F) + f(n-2, F)
}
```

 $\Theta(n)$ 

Memoization



# แบบมีกับไม่มี memoization



# เติมตารางอีกแบบ : Bottom-Up

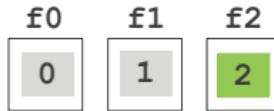
	0	1	2	3	4	5
F	0	1	1	2	3	5

```
f( n ) {
    F = new array[0..n]
    F[0] = 0; F[1] = 1
    for (i = 2; i <= n; i++) {
        F[i] = F[i - 1] + F[i - 2]
    }
    return F[n]
}
```

$\Theta(n)$

bottom-up

# ลดขนาดของตารางเหลือแค่ 3 ตัว

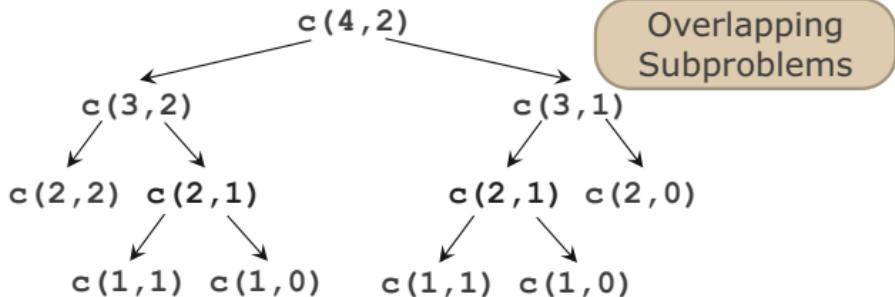


```
f( n ) {  
    f0 = 0, f1 = 1, f2 = 1  
    for (i = 2; i <= n; i++) {  
        f2 = f0 + f1  
        f0 = f1  
        f1 = f2  
    }  
    return f2  
}
```

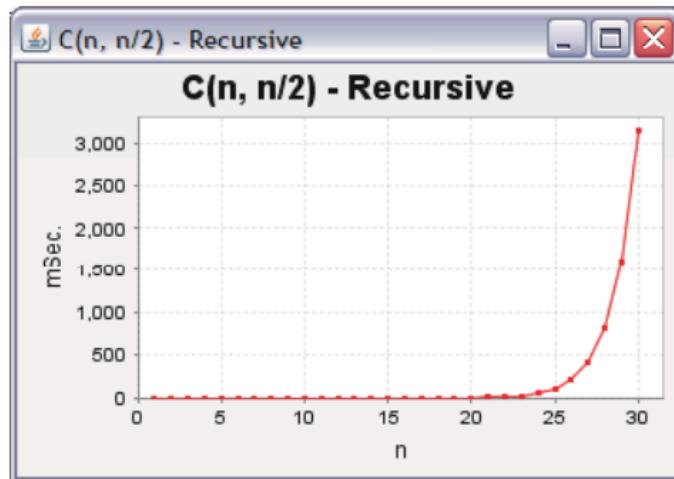
# C(n,k) : n เลือก k

$$C(n, k) = \begin{cases} C(n-1, k) + C(n-1, k-1) & \text{if } 0 < k < n \\ 1 & \text{if } k = 0 \text{ or } k = n \\ 0 & \text{otherwise} \end{cases}$$

```
c(n, k) {
    if (k == 0 OR k == n) return 1
    if (k < 0 OR k > n) return 0
    return c(n-1, k) + c(n-1, k-1)
}
```



# $C(n, k)$ : เวลาการทำงาน



# C(n,k) : Top-down + Memoization

c(10,5)

ทำ 503 ครั้ง

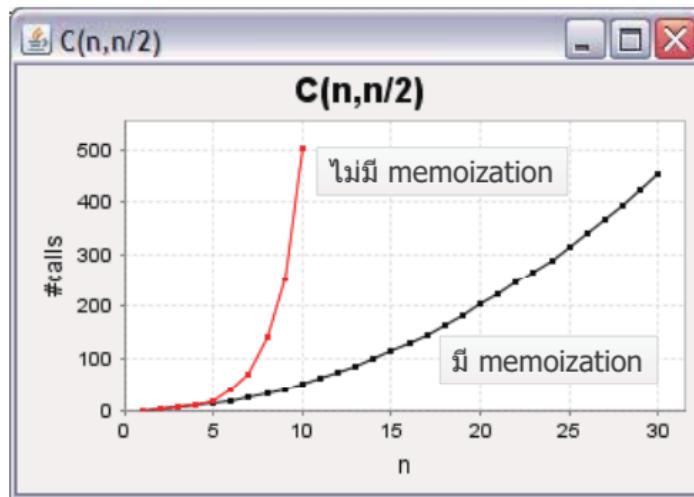
```
c(n, k) {
    if (k == 0 OR k == n) return 1
    if (k < 0 OR k > n) return 0
    return c(n-1, k) + c(n-1, k-1)
}
```

ทำ 51 ครั้ง

c(10, 5, new array[1..10][1..5])

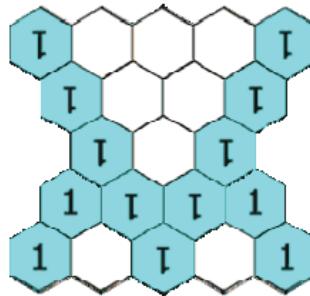
```
c(n, k, C[1..n][1..k]) {
    if (k == 0 OR k == n) return 1
    if (k < 0 OR k > n) return 0
    if (C[n][k] > 0) return C[n][k]
    return C[n][k] = c(n-1, k, C) + c(n-1, k-1, C)
}
```

# แบบมีกับไม่มี memoization

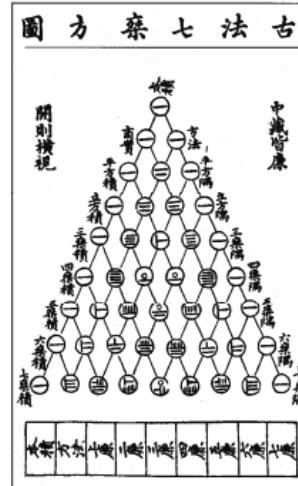


# $C(n, k)$ : Bottom-up

Pascal's Triangle



Bottom = หากำตอบของกรณีเล็ก ๆ  
 Up = หากำตอบของกรณีใหญ่ขึ้น  
 โดยใช้หากำตอบของกรณีเล็กที่รู้แล้ว



# Pascal's Triangle : เขียนอีกแบบ

		k	3	4	5	6
		0	1	2	3	4
n	0	1				
	1	1	1			
	2	1	2	1		
	3	1	3	3	1	
	4	1	4	6	4	1
	5	1	5	10	10	5
	6	1	6	15	20	15

$C(6, 2) = 15$

# C(n, k) : Bottom-up

$$C(n, k) = \begin{cases} C(n-1, k) + C(n-1, k-1) & \text{if } 0 < k < n \\ 1 & \text{if } k = 0 \text{ or } k = n \\ 0 & \text{otherwise} \end{cases}$$

k

	0	1	2
0	1		
1	1	1	
2	1	2	1
n	3	1 3 3	
4	1 4 6		
5	1 5 10		
6	1 6 15		

ต้องการ C(6, 2)

# C(n, k) : Bottom-up

```
c(n, k) {  
    C = new array[0..n][0..k]  
    for (i = 0; i <= n; i++) C[i][0] = 1  
    for (i = 0; i <= k; i++) C[i][i] = 1  
    for (i = 2; i <= n; i++)  
        for (j = 1; j <= k && j < i; j++)  
            C[i][j] = C[i - 1][j] + C[i - 1][j - 1]  
    return C[n][k];  
}
```

$\Theta(nk)$

n \ k	0	1	2
0	1		
1	1	1	
2	1	2	1
3	1	3	3
4	1	4	6
5	1	5	10
6	1	6	15

# Top-Down vs. Bottom-Up

## ❖ Top-Down

- ❖ แบ่งปัญหาใหญ่เป็นปัญหาย่อย
- ❖ หาคำตอบของปัญหาย่อย
- ❖ นำคำตอบย่อย ๆ มารวมเป็นคำตอบของปัญหาใหญ่
- ❖ มักเขียนแบบ recursive
- ❖ แก้ปัญหาย่อยเท่าที่จำเป็น (แต่อาจแก้ซ้ำ แก้แล้วแก้อีก)
- ❖ ลดการแก้ปัญหาย่อยซ้ำด้วย memoization

## ❖ Bottom-Up

- ❖ เริ่มหาคำตอบของปัญหาราก ๆ
- ❖ นำคำตอบของปัญหารากมาหาคำตอบของปัญหาใหญ่ขึ้น
- ❖ มักเขียนแบบวนว่น
- ❖ แก้ปัญหาย่อย ๆ ทุกปัญหา ปัญหาระหว่าง

# Dynamic Programming

- ❖ ทำงานแบบ bottom-up ด้วยการแก้ปัญหาย่อย จำคำตอบที่ได้ เพื่อนำไปใช้แก้ปัญหาใหญ่ขึ้น
- ❖ มักใช้กับ optimization problems
  - ❖ การหาเส้นทางสั้นสุด
  - ❖ การหาลำดับย่อยร่วมกันที่ยาวสุด
  - ❖ การเลือกของที่ได้มูลค่ารวมสูงสุด
  - ❖ การจัดเก็บข้อมูลในต้นไม้ค้นหาให้ค้นได้เร็วสุด
  - ❖ การหาลำดับการคุณเมทริกซ์ เพื่อให้คุณได้เร็วสุด
  - ❖ ...



Richard Bellman นักคณิตศาสตร์  
ผู้คิดวิธี Dynamic programming  
ในปี ค.ศ. 1953

# Longest Common Subsequence

แฟ้มสองแฟ้มเหมือนกัน (ต่างกัน) ตรงไหนบ้าง

The screenshot shows the TextDiff application interface comparing two files:

- Left File:** C:\Program Files\Borland\Delphi7\Files\Diff\TextDiff\Backup\DiffUnit.pas
- Right File:** C:\Program Files\Borland\Delphi7\Files\Diff\TextDiff\Source\DiffUnit.pas

The code snippets are as follows:

```

Left File (Line numbers 398-406):
398     end;
399     //update current vector ...
400     diagVecB[k] := x;
401     {$IFDEF DIFF_PROGRESS}
402     if x < fBackwardX then fBackw
403     {$ENDIF}
404
405     //check if midpoint reached
406     if not odd(Delta) and (k >=

```

```

Right File (Line numbers 442-451):
442     end;
443     //update current vector ...
444     diagVecB[k] := curr1;
445
446     {$IFDEF DIFF_PROGRESS}
447     if curr1 < fBackwardX then
448     {$ENDIF}
449
450     //check if a crossover point
451     if not odd(Delta) and (k >=

```

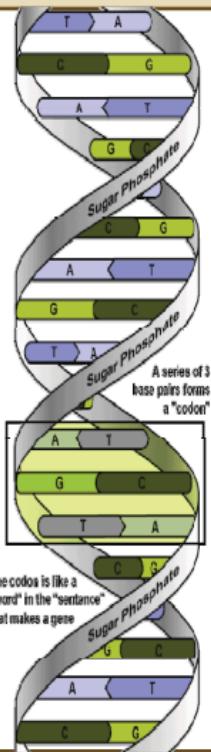
Annotations in the left file highlight lines 400, 402, 403, 405, and 406. Annotations in the right file highlight lines 444, 447, 448, 450, and 451.

At the bottom of the interface, a status bar displays: "93 lines added, 134 lines modified, 19 lines deleted."

# Longest Common Subsequence

วัดความคล้ายของ DNA sequences

ACCGGTGAGTGCCTGGAAAGCCGGCCGAA  
GTCGTTCTCGGAATGCCGTTGCTCTGTAAA.



# Longest Common Subsequence

❖  $X = \langle H, E, L, L, O \rangle$  มี subsequences

- ❖  $\langle H, E, L, L, O \rangle \rightarrow \langle H, E \rangle$
- ❖  $\langle H, E, L, L, O \rangle \rightarrow \langle H, E, O \rangle$
- ❖  $\langle H, E, L, L, O \rangle \rightarrow \langle \rangle$
- ❖ ...

❖  $Y = \langle H, E, R, O \rangle$  มี subsequences

- ❖  $\langle H, E, R, O \rangle \rightarrow \langle H, E \rangle$
- ❖  $\langle H, E, R, O \rangle \rightarrow \langle H, R \rangle$
- ❖  $\langle H, E, R, O \rangle \rightarrow \langle H, E, O \rangle$
- ❖ ...

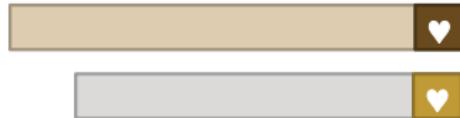
❖ common subsequence( $X, Y$ )

❖ longest common subsequence( $X, Y$ )

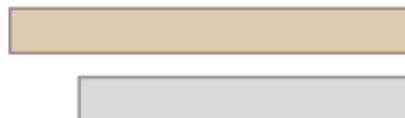
## นิยาม

- ❖  $X = \langle x_1, x_2, x_3, \dots, x_m \rangle$
- ❖  $Y = \langle y_1, y_2, y_3, \dots, y_n \rangle$
- ❖  $X_i = \langle x_1, x_2, \dots, x_i \rangle$
- ❖  $Y_j = \langle y_1, y_2, \dots, y_j \rangle$
- ❖  $\text{LCS}(X_i, Y_j)$  :  
longest common subsequence ของ  $X_i, Y_j$
- ❖  $\text{LCS}(X, Y) = \text{LCS}(X_m, Y_n)$
- ❖  $L(i, j)$  : ความยาวของ  $\text{LCS}(X_i, Y_j)$

# คำตอบไขญี่ได้จากคำตอบย่อๆ

LCS(  )

=

LCS(  ) + 

$$\begin{aligned} \text{LCS}(\text{HELLO}, \text{HERO}) &= \text{LCS}(\text{HELL}, \text{HER}) + \text{O} \\ &= \text{HE} + \text{O} \\ &= \text{HEO} \end{aligned}$$

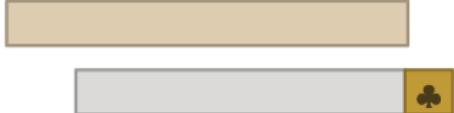
## คำตอบในญี่ได้จากคำตอบย่ออย ๑

LCS(  ) LCS(COP, CEO)

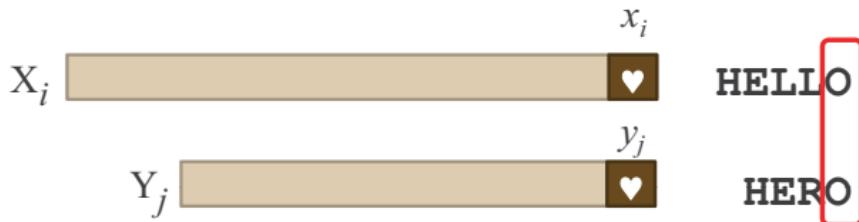
=

=

LCS(  ) LCS(COP, CE)

LCS(  ) LCS(CO, CEO)

# Recurrence ของ $L(i, j)$

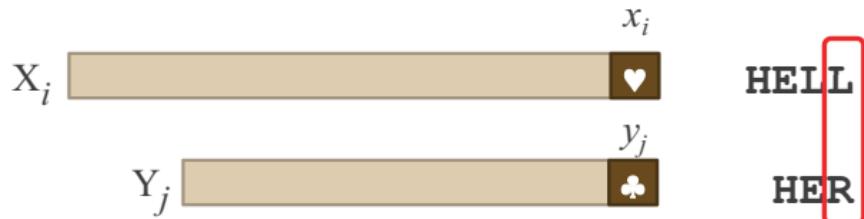


$$L(i, j) = L(i - 1, j - 1) + 1 \quad \text{if } x_i = y_j$$

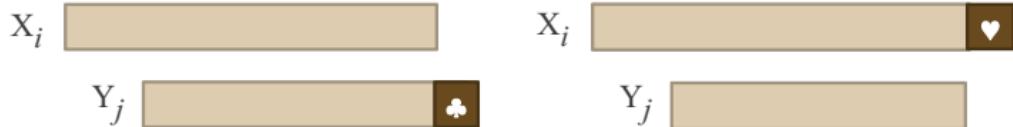
↑                        ↑

ความยาวของ  $LCS(X_{i-1}, Y_{j-1})$           ตัวขวาที่เท่ากัน 1 ตัว

# Recurrence ของ $L(i, j)$



$$L(i, j) = \max( L(i - 1, j) , L(i, j - 1) ) \quad \text{if } x_i \neq y_j$$



# Recurrence ของ $L(i, j)$

$X_i$  [  $x_i$  ]

$Y_j$  [  $y_j$  ]

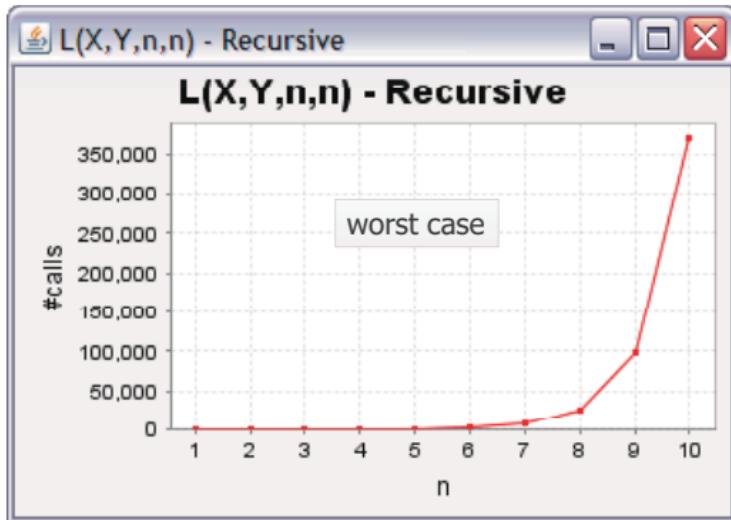
$$L(i, j) = \begin{cases} L(i-1, j-1) + 1 & \text{if } x_i = y_j \\ \max(L(i-1, j), L(i, j-1)) & \text{if } x_i \neq y_j \\ 0 & \text{if } i = 0 \text{ or } j = 0 \end{cases}$$

## LCS : Top-down

$$L(i, j) = \begin{cases} L(i-1, j-1) + 1 & \text{if } x_i = y_j \\ \max(L(i-1, j), L(i, j-1)) & \text{if } x_i \neq y_j \\ 0 & \text{if } i = 0 \text{ or } j = 0 \end{cases}$$

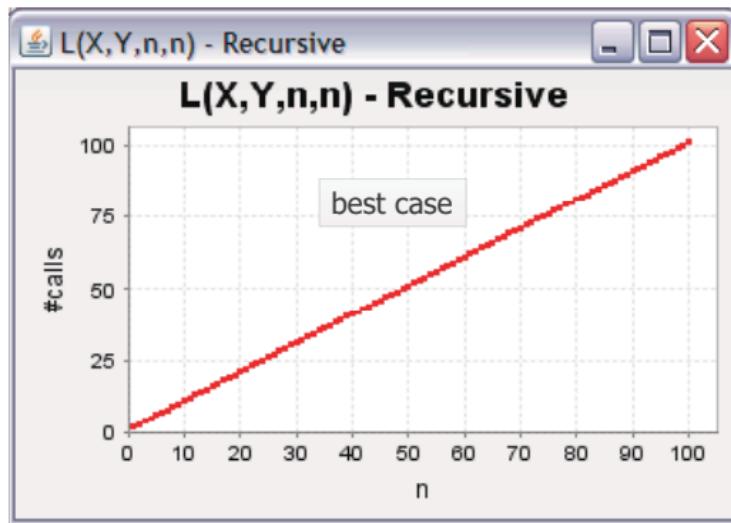
```
L(x[1..m], y[1..n]) {
    return L(x, y, m, n)
}
L(x[1..m], y[1..n], i, j) {
    if (i == 0 OR j == 0) return 0
    if (x[i] == y[j]) {
        return L(x, y, i-1, j-1) + 1
    } else {
        return max( L(x, y, i-1, j),
                    L(x, y, i, j-1) )
    }
}
```

# หาความยาว LCS : เวลาการทำงาน



ลักษณะข้อมูลของกรณี worst case และ best case เป็นอย่างไร ?  
 กรณี best case มีประสิทธิภาพเชิงเวลาอย่างไร ?  
 ลองปรับด้วย memoization จะได้ผลอย่างไร ?

# LCS : best cases

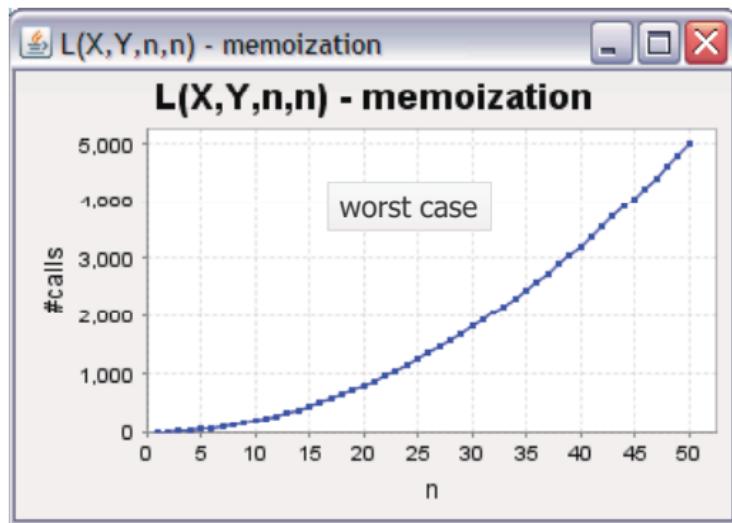


worst case เมื่อสองลำดับไม่เหมือนกันเลย  
best case เมื่อสองลำดับเหมือนกันหมด

# Top-down + Memoization

$$L(i, j) = \begin{cases} L(i-1, j-1) + 1 & \text{if } x_i = y_j \\ \max(L(i-1, j), L(i, j-1)) & \text{if } x_i \neq y_j \\ 0 & \text{if } i = 0 \text{ or } j = 0 \end{cases}$$

# LCS : memoization



# LCS : Bottom-up

$$L(i, j) = \begin{cases} L(i-1, j-1) + 1 & \text{if } x_i = y_j \\ \max(L(i-1, j), L(i, j-1)) & \text{if } x_i \neq y_j \\ 0 & \text{if } i = 0 \text{ or } j = 0 \end{cases}$$

		H	E	L	L	O
		0	0	0	0	0
		0	1	1	1	1
		0	1	2	2	2
		0	1	2	2	2
		0	1	2	2	3

# LCS : Dynamic Prog.

$$L(i, j) = \begin{cases} L(i-1, j-1) + 1 & \text{if } x_i = y_j \\ \max(L(i-1, j), L(i, j-1)) & \text{if } x_i \neq y_j \\ 0 & \text{if } i = 0 \text{ or } j = 0 \end{cases}$$

```
LCS_Length(x[1..m], y[1..n]) {  
    L = new array[0..m][0..n]  
    for (i = 0; i <= m; i++) L[i][0] = 0  
    for (j = 0; j <= n; j++) L[0][j] = 0  
    for (i = 1; i <= m; i++)  
        for (j = 1; j <= n; j++)  
            if (x[i] == y[j])  
                L[i][j] = L[i - 1][j - 1] + 1  
            else  
                L[i][j] = max(L[i - 1][j], L[i][j - 1])  
    return L;  
}
```



$\Theta(nm)$

# ต้องการ LCS : จำผลการตัดสินใจ

$$L(i, j) = \begin{cases} L(i-1, j-1) + 1 & \text{if } x_i = y_j \\ \max(L(i-1, j), L(i, j-1)) & \text{if } x_i \neq y_j \\ 0 & \text{if } i = 0 \text{ or } j = 0 \end{cases}$$

	j	H	E	L	L	O
i		0	0	0	0	0
H	0	1	1	1	1	1
E	0	1	2	2	2	2
R	0	1	2	2	2	2
O	0	1	2	2	2	3

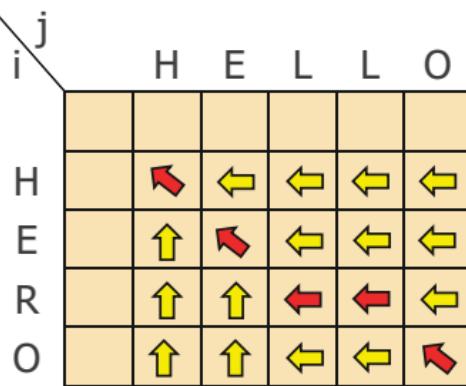
# จำผลการตัดสินใจ

```
LCS (x[1..m], y[1..n]) {  
    L = new array[0..m][0..n]  
    D = new array[1..m][1..n]  
    for (i = 0; i <= m; i++) L[i][0] = 0  
    for (j = 0; j <= n; j++) L[0][j] = 0  
    for (i = 1; i <= m; i++)  
        for (j = 1; j <= n; j++)  
            if (x[i] == y[j])  
                L[i][j] = L[i - 1][j - 1] + 1  
                D[i][j] = ↘  
            else  
                if (L[i - 1][j] > L[i][j - 1])  
                    L[i][j] = L[i - 1][j]  
                    D[i][j] = ↑  
                else  
                    L[i][j] = L[i][j - 1]  
                    D[i][j] = ←
```

# ใช้ผลการตัดสินใจสร้างคำตอบ

```

lcs = an empty sequence
i = n, j = m
while (i > 0 AND j > 0) {
    switch( D[i][j] ) {
        case : ↗
            lcs = x[i] + lcs
            i--; j--
        case : ←
            j--
        case : ↑
            i--
    }
}
return lcs
}
    
```



# ต้องการประยัด ไม่จำผลการตัดสินใจ

$$L(i, j) = \begin{cases} L(i-1, j-1) + 1 & \text{if } x_i = y_j \\ \max(L(i-1, j), L(i, j-1)) & \text{if } x_i \neq y_j \\ 0 & \text{if } i = 0 \text{ or } j = 0 \end{cases}$$

		j	H	E	L	L	O
i		0	0	0	0	0	0
H	0	1	1	1	1	1	
E	0	1	2	2	2	2	
R	0	1	2	2	2	2	
O	0	1	2	2	2	2	3

# สร้าง LCS ได้จากตาราง L

$$L(i, j) = \begin{cases} L(i-1, j-1) + 1 & \text{if } x_i = y_j \\ \max(L(i-1, j), L(i, j-1)) & \text{if } x_i \neq y_j \\ 0 & \text{if } i = 0 \text{ or } j = 0 \end{cases}$$

		H	E	L	L	O
		0	0	0	0	0
		0	1	1	1	1
		0	1	2	2	2
R		0	1	2	2	2
O		0	1	2	2	3

# สร้าง LCS ได้จากตาราง L

$$L(i, j) = \begin{cases} L(i-1, j-1) + 1 & \text{if } x_i = y_j \\ \max(L(i-1, j), L(i, j-1)) & \text{if } x_i \neq y_j \\ 0 & \text{if } i = 0 \text{ or } j = 0 \end{cases}$$

		j	H	E	L	L	O
i		0	0	0	0	0	0
H	0	1	1	1	1	1	1
E	0	1	2	2	2	2	2
R	0	1	2	2	2	2	2
O	0	1	2	2	2	2	3

# สร้าง LCS ได้จากตาราง L

$$L(i, j) = \begin{cases} L(i-1, j-1) + 1 & \text{if } x_i = y_j \\ \max(L(i-1, j), L(i, j-1)) & \text{if } x_i \neq y_j \\ 0 & \text{if } i = 0 \text{ or } j = 0 \end{cases}$$

		j					
		H	E	L	L	O	
i		0	0	0	0	0	0
	H	0	1	1	1	1	1
	E	0	1	2	2	2	2
	R	0	1	2	2	2	2
	O	0	1	2	2	2	3

# สร้าง LCS ได้จากตาราง L

$$L(i, j) = \begin{cases} L(i-1, j-1) + 1 & \text{if } x_i = y_j \\ \max(L(i-1, j), L(i, j-1)) & \text{if } x_i \neq y_j \\ 0 & \text{if } i = 0 \text{ or } j = 0 \end{cases}$$

		j	H	E	L	L	O
i		0	0	0	0	0	0
H	0	1	1	1	1	1	1
E	0	1	2	2	2	2	2
R	0	1	2	2	2	2	2
O	0	1	2	2	2	2	3

# สร้าง LCS ได้จากตาราง L

$$L(i, j) = \begin{cases} L(i-1, j-1) + 1 & \text{if } x_i = y_j \\ \max(L(i-1, j), L(i, j-1)) & \text{if } x_i \neq y_j \\ 0 & \text{if } i = 0 \text{ or } j = 0 \end{cases}$$

	j	H	E	L	L	O
i	0	0	0	0	0	0
H	0	1	1	1	1	1
E	0	1	2	2	2	2
R	0	1	2	2	2	2
O	0	1	2	2	2	3

# สร้าง LCS ได้จากตาราง L

$$L(i, j) = \begin{cases} L(i-1, j-1) + 1 & \text{if } x_i = y_j \\ \max(L(i-1, j), L(i, j-1)) & \text{if } x_i \neq y_j \\ 0 & \text{if } i = 0 \text{ or } j = 0 \end{cases}$$

		j	H	E	L	L	O
i		0	0	0	0	0	0
H	0	1	1	1	1	1	
E	0	1	2	2	2	2	
R	0	1	2	2	2	2	
O	0	1	2	2	2	3	

# LCS\_Soln(x, y, L)

```

LCS_Soln(x[1..n], y[1..m], L[0..n][0..m]) {
    lcs = an empty sequence
    i = n, j = m
    while (i > 0 AND j > 0) {
        if (x[i] == y[j]) {
            lcs = x[i] + lcs
            i--; j--
        } else {
            if (L[i][j-1] > L[i-1][j])
                j--
            else
                i--
        }
    }
    return lcs
}

```

สามารถหา LCS ได้ด้วยเนื้อที่  $O(n + m)$

D. Hirschberg "A linear space algorithm for computing maximal common subsequences"

# สะกดผิด : หาคำใกล้เคียง

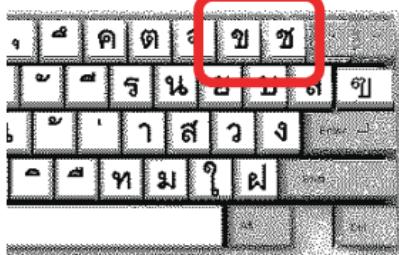
พัชราภา

ค้น

edit distance = 1

edit distance = 2

พัชราภา



พัชรนภา,

พัชราภร,

พิราภา,

ภัสรภา,

วัชราภา

พัชรภา

พัชราภา

ภัตราภา

ภัสสรภา

# Minimum Edit Distance

- ❖ **เปลี่ยน HELLO ให้เป็น HERO เช่น**
  - ❖ HELLO → ELLO → LLO → LO → RO → ERO → HERO
  - ❖ HELLO → HELO → HERO
- ❖ **ให้การแก้ไขสตริงประกอบด้วย**
  - ❖ การลบอักขระหนึ่งตัว
  - ❖ การเพิ่มอักขระหนึ่งตัว
  - ❖ การแทนอักขระหนึ่งตัวด้วยอักขระอีกตัว
- ❖ **การแก้ไขมีต้นทุน  $c_D$ ,  $c_I$ ,  $c_S$**
- ❖ **Edit distance** คือต้นทุนการแก้ไขเพื่อเปลี่ยนสตริง
- ❖ **ปัญหา : เปลี่ยนสตริง  $s$  ให้เป็น  $t$  อย่างไรให้มี minimum edit distance**

ลองทำเอง  
คล้าย ๆ LCS

# ลักษณะของปัญหา

## ❖ overlapping subproblems :

- ❖ ต้องแก้ปัญหาย่อย ๆ ข้า ๆ หลายครั้ง การจำคำตอบจึงประยุ้ดเวลาการทำงานได้มาก
- ❖ ต้องมีจำนวนปัญหาย่อยทั้งหมดไม่มาก

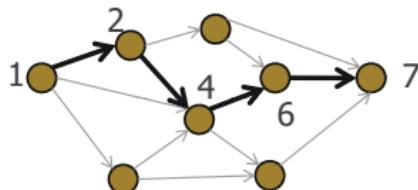
## ❖ optimal substructures (principle of optimality) :

- ❖ คำตอบที่ดีสุดของปัญหาใหญ่ได้มาจากการคำตอบที่ดีสุดของปัญหาที่เล็กกว่า
- ❖ ทำให้เขียน recurrence ของคุณภาพของคำตอบได้
- ❖ คิด : ใช้คำตอบดีสุดของปัญหาเล็กได และเลือกอย่างไร

$$L(i, j) = \begin{cases} L(i-1, j-1) + 1 & \text{if } x_i = y_j \\ \max(L(i-1, j), L(i, j-1)) & \text{if } x_i \neq y_j \\ 0 & \text{if } i = 0 \text{ or } j = 0 \end{cases}$$

# Optimal Substructures

$\text{LCS}(\text{"HELLO"}, \text{"HERO"})$  คือ  $\text{LCS}(\text{"HELL"}, \text{"HER"})$  ต่อด้วย "O"



ถ้าทางเดินสั้นสุดจาก 1 ไป 7 คือ  $1 \rightarrow 2 \rightarrow 4 \rightarrow 6 \rightarrow 7$   
แสดงว่า

ทางจาก 1 ไป 4 ชึ้งคือ  $1 \rightarrow 2 \rightarrow 4$  ต้องสั้นสุดด้วย

ทางจาก 1 ไป 6 ชึ้งคือ  $1 \rightarrow 2 \rightarrow 4 \rightarrow 6$  ต้องสั้นสุดด้วย

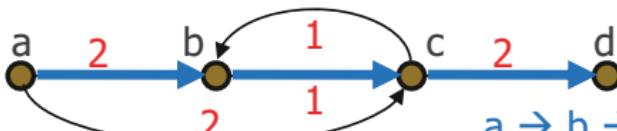
ทางจาก 2 ไป 6 ชึ้งคือ  $2 \rightarrow 4 \rightarrow 6$  ต้องสั้นสุดด้วย

ทางจาก 2 ไป 7 ชึ้งคือ  $2 \rightarrow 4 \rightarrow 6 \rightarrow 7$  ต้องสั้นสุดด้วย

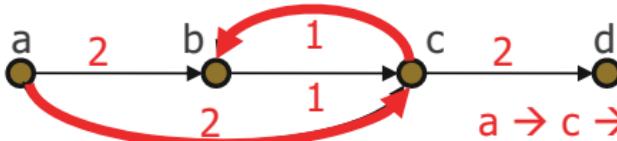
ปัญหา optimization ไม่จำเป็นต้องมี  
optimal substructure เสมอไป

# Longest Simple Path

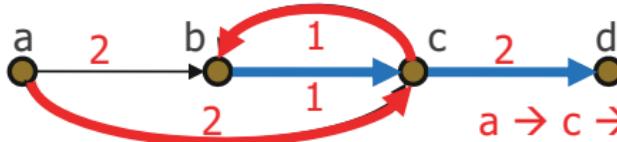
Simple path คือ path ที่ไม่ผ่านปมซ้ำ



$a \rightarrow b \rightarrow c \rightarrow d$  ยาวสุด  
แต่  $a \rightarrow b$  ไม่ยาวสุด



$a \rightarrow c \rightarrow b$  ยาวกว่า



$a \rightarrow c \rightarrow b \rightarrow c \rightarrow d$   
ยาวกว่า แต่ไม่ใช้ simple path

ไม่มี optimal substructure  
ใช้ dynamic programming ไม่ได้

# 0/1 Knapsack

- ❖ ของ  $n$  ชิ้นมีหมายเลข :  $1, 2, 3, \dots, n$
- ❖ แต่ละชิ้นหนัก :  $w_1, w_2, w_3, \dots, w_n$
- ❖ แต่ละชิ้นมีมูลค่า :  $v_1, v_2, v_3, \dots, v_n$
- ❖ ถุงเป้หนึ่งใบจุของได้นักไม่เกิน  $W$
- ❖ ปัญหา : จงเลือกของใส่ถุง เพื่อให้
  - ❖ ถุงไม่ขาด
  - ❖ ได้มูลค่ารวมมากสุด



2.5Kg.  
\$400



0.7Kg.  
\$150



4Kg.

1.4Kg.  
\$50



2.8Kg.  
\$350



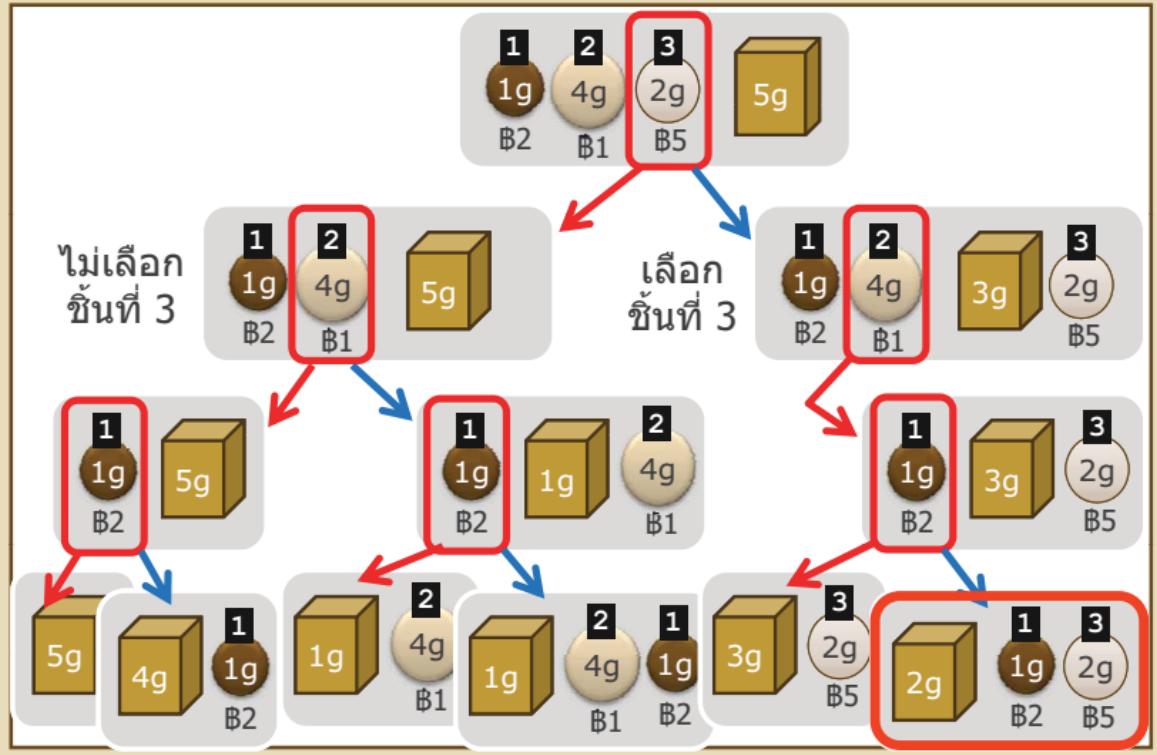
# 0/1 Knapsack

- ❖ ของ  $n$  ชิ้นมีหมายเลข :  $1, 2, 3, \dots, n$
- ❖ แต่ละชิ้นหนัก :  $w_1, w_2, w_3, \dots, w_n$
- ❖ แต่ละชิ้นมีมูลค่า :  $v_1, v_2, v_3, \dots, v_n$
- ❖ ถุงเป้หนึ่งใบจุของได้หนักไม่เกิน  $W$
- ❖ หา  $\langle x_1, x_2, x_3, \dots, x_n \rangle$ ,  $x_k = 0$  หรือ 1

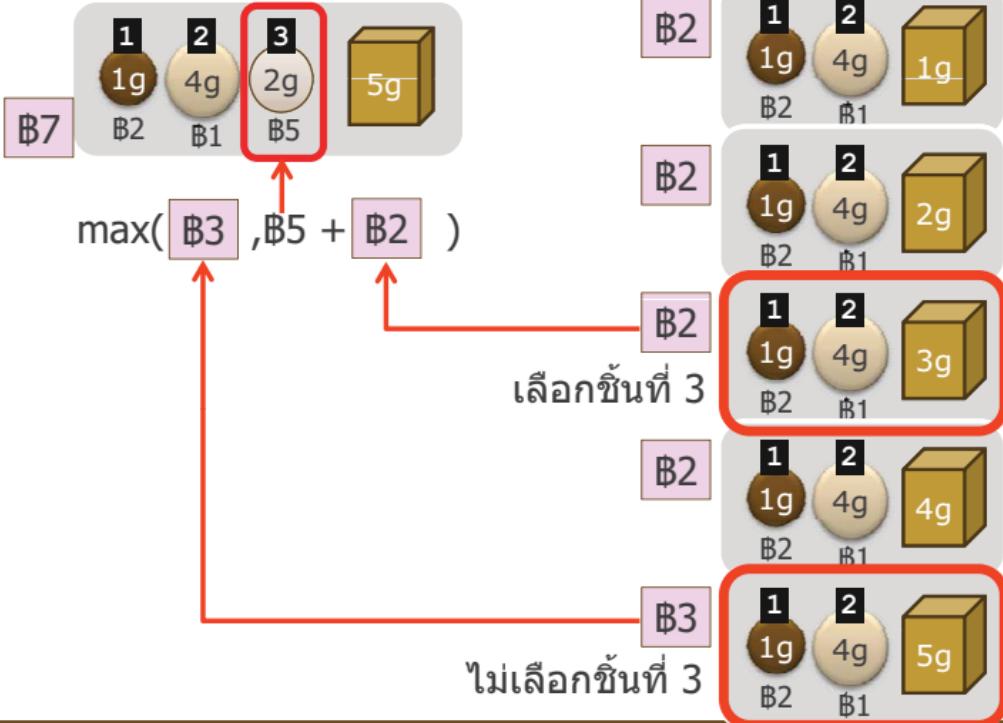
$$\begin{aligned} & \text{maximize} \quad \sum_{k=1}^n x_k v_k \\ & \text{subject to} \quad \sum_{k=1}^n x_k w_k \leq W \end{aligned}$$

$$x_k \in \{0,1\}$$

# ແບ່ງປັນຫາໃຫຍ່ເປັນປັນຫາຍ່ອຍ



# หาคำตอบให้ญี่จากคำตอบเล็ก



# ปัญหาใหญ่ : ปัญหาย่อย

- ❖ **LCS** :  $L(i, j)$ 
  - ❖ ปัญหาใหญ่หรือเล็กขึ้นกับความยาวของลำดับ  $X_i$  และ  $Y_j$
- ❖ **Edit distance** :  $D(i, j)$
- ❖ **Knapsack** :  $V(?)$ 
  - ❖ ปัญหาใหญ่หรือเล็กขึ้นกับจำนวนของ และน้ำหนักที่ถุงรับได้
  - $V(i, j)$  : มูลค่ารวมสูงสุดในการ
    - ❖ เลือกของชิ้นที่ 1, 2, 3, ...,  $i$
    - ❖ ใส่ถุงเป็ที่รับน้ำหนักได้ไม่เกิน  $j$

เขียน recurrence ของ  $V(i, j)$

# เขียน recurrence $V(i, j)$



$$V(3, 5) = \max(\quad, \quad)$$

$$V(3-1, 5)$$

$$v_3 + V(3-1, 5-w_3)$$

ไม่เลือก  
ชิ้นที่ 3



เลือก  
ชิ้นที่ 3



$V(i, j)$  แทนมูลค่าของการเลือกที่ดีสุดเมื่อ  
มีของให้เลือกชิ้นที่  $1, 2, \dots, i$  ใส่ถุงเป้าที่จุได้นัก  $j$

$$V(i, j) = \begin{cases} \max(V(i-1, j), v_i + V(i-1, j - w_i)) & \text{if } i > 0 \text{ and } j \geq w_i \\ V(i-1, j) & \text{if } j < w_i \\ 0 & \text{if } i = 0 \text{ or } j = 0 \end{cases}$$

# Knapsack : Top-down

$$V(i, j) = \begin{cases} \max(V(i-1, j), v_i + V(i-1, j - w_i)) & \text{if } i > 0 \text{ and } j \geq w_i \\ V(i-1, j) & \text{if } j < w_i \\ 0 & \text{if } i = 0 \text{ or } j = 0 \end{cases}$$

```

V( v[1..n] , w[1..n] , w ) {
    return V(v, w, n, W)
}
V( v[1..n] , w[1..n] , i , j ) {
    if (i == 0 OR j == 0) return 0
    if (j < w[i]) {
        return V(v, w, i-1, j)
    } else {
        return max( V(v, w, i-1, j),
                    V(v, w, i-1, j-w[i]) + v[i] )
    }
}

```

## Top-down + Memoization

$$V(i, j) = \begin{cases} \max(V(i-1, j), v_i + V(i-1, j - w_i)) & \text{if } i > 0 \text{ and } j \geq w_i \\ V(i-1, j) & \text{if } j < w_i \\ 0 & \text{if } i = 0 \text{ or } j = 0 \end{cases}$$

```
V(v[1..n], w[1..n], i, j, M[1..n][1..W]) {
    if (i == 0 || j == 0) return 0
    if (M[i][j] > 0) return M[i][j]
    if (j < w[i]) {
        return M[i][j] = V(v, w, i-1, j)
    } else {
        return M[i][j] = max( V(v, w, i-1, j),
                            V(v, w, i-1, j-w[i]) )
    }
}
```

# 0/1 Knapsack : Bottom-Up

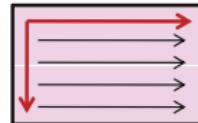
$$V(i, j) = \begin{cases} \max(V(i-1, j), v_i + V(i-1, j - w_i)) & \text{if } i > 0 \text{ and } j \geq w_i \\ V(i-1, j) & \text{if } j < w_i \\ 0 & \text{if } i = 0 \text{ or } j = 0 \end{cases}$$

		j	0	1	2	3	4	5	6	7	8	9	10
i		0	0	0	0	0	0	0	0	0	0	0	0
v <sub>i</sub>	w <sub>i</sub>	0	0	0	0	0	0	0	0	0	0	0	0
20	2	1	0	0	20	20	20	20	20	20	20	20	20
30	2	2	0	0	30	30	50	50	50	50	50	50	50
66	3	3	0	0	30	66	66	96	96	116	116	116	116
40	4	4	0	0	30	66	66	96	96	116	116	136	136
60	5	5	0	0	30	66	66	96	96	116	126	136	156

$$W = 10$$

# 0/1 Knapsack : Bottom-Up

```
knapsack_Value(v[1..n], w[1..n], W) {  
    V = new array[0..n][0..W]  
    for (i = 0; i <= n; i++) V[i][0] = 0  
    for (j = 0; j <= W; j++) V[0][j] = 0  
    for (i = 1; i <= n; i++)  
        for (j = 1; j <= W; j++)  
            if (j < w[i])  
                V[i][j] = V[i-1][j]  
            else  
                V[i][j] = max(V[i-1][j]), v[i]+V[i-1][j-w[i]])  
    return V  
}
```



$\Theta(nW)$

$$V(i, j) = \begin{cases} \max(V(i-1, j), v_i + V(i-1, j - w_i)) & \text{if } i > 0 \text{ and } j \geq w_i \\ V(i-1, j) & \text{if } j < w_i \\ 0 & \text{if } i = 0 \text{ or } j = 0 \end{cases}$$

# อยากรู้เลือกอะไร : จำผลการตัดสินใจ

$$V(i, j) = \begin{cases} \max(V(i-1, j), v_i + V(i-1, j - w_i)) & \text{if } i > 0 \text{ and } j \geq w_i \\ V(i-1, j) & \text{if } j < w_i \\ 0 & \text{if } i = 0 \text{ or } j = 0 \end{cases}$$

		j	0	1	2	3	4	5
i		0	0	0	0	0	0	0
$v_i$	$w_i$	0	0	0	20	20	20	20
20	2	1	0	0	20	20	20	20
30	2	2	0	0	30	30	50	50
25	3	3	0	0	30	30	50	55

$$W = 5$$

# ใช้ผลการตัดสินใจสร้างคำตอบ

		j	0	1	2	3	4	5
i	$v_i$	$w_i$	0	0	0	0	0	0
20	2	1	0	✗	✓	✓	✓	✓
30	2	2	0	✗	✓	✓	✓	✓
25	3	3	0	✗	✗	✗	✗	✓

# 0/1 Knapsack : จําการตัดสินใจ

```

knapsack(v[1..n], w[1..n], W) {
    V = new array[0..n][0..W]
    X = new array[1..n][1..W]
    for (i = 0; i <= n; i++) V[i][0] = 0
    for (j = 0; j <= W; j++) V[0][j] = 0
    for (i = 1; i <= n; i++)
        for (j = 1; j <= W; j++)
            if (j < w[i])
                V[i][j] = V[i-1][j]; X[i][j] = ✗
            else
                if (V[i-1][j]) > v[i]+V[i-1][j - w[i]] ) {
                    V[i][j] = V[i-1][j]
                    X[i][j] = ✗
                } else {
                    V[i][j] = v[i] + V[i-1][j - w[i]]
                    X[i][j] = ☑
                }
}

```

# 0/1 Knapsack : จําการตัดสินใจ

```
S = an empty set
i = n; j = W
while (i > 0 AND j > 0) {
    if ( X[i][j] ==  ) {
        S.add(i);
        j = j - w[i];
    }
    i--;
}
return S;
}
```

	i \ j	0	1	2	3	4	5
v <sub>i</sub>	0	0	0	0	0	0	0
w <sub>i</sub>	20	2	1	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
	30	2	2	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
	25	3	3	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

# ต้องการประยุ้ด ไม่จำเพาะการตัดสินใจ

$$V(i, j) = \begin{cases} \max(V(i-1, j), v_i + V(i-1, j-w_i)) & \text{if } i > 0 \text{ and } j \geq w_i \\ V(i-1, j) & \text{if } j < w_i \\ 0 & \text{if } i = 0 \text{ or } j = 0 \end{cases}$$

	$i$	0	1	2	3	4	5	6	7	8	9	10
$v_i$	$w_i$	0	0	0	0	0	0	0	0	0	0	0
20	2	1	0	0	20	20	20	20	20	20	20	20
30	2	2	0	0	30	30	50	50	50	50	50	50
66	3	3	0	0	30	66	66	96	96	116	116	116
40	4	4	0	0	30	66	66	96	96	116	116	136
60	5	5	0	0	30	66	66	96	96	116	126	136

$i = 0 \text{ or } j = 0 ? \rightarrow \text{false}$

$j < w_i ? \rightarrow \text{false}$

$V(i, j) = v_i + V(i-1, j-w_i) ? \rightarrow \text{true} \rightarrow \text{เลือกชั้นที่ 5}$

# แล้วเลือกของชิ้นใดบ้าง ?

$$V(i, j) = \begin{cases} \max(V(i-1, j), v_i + V(i-1, j-w_i)) & \text{if } i > 0 \text{ and } j \geq w_i \\ V(i-1, j) & \text{if } j < w_i \\ 0 & \text{if } i = 0 \text{ or } j = 0 \end{cases}$$

		j	0	1	2	3	4	5	6	7	8	9	10
i			0	0	0	0	0	0	0	0	0	0	0
$v_i$	$w_i$	0	0	0	0	0	0	0	0	0	0	0	0
20	2	1	0	0	20	20	20	20	20	20	20	20	20
30	2	2	0	0	30	30	50	50	50	50	50	50	50
66	3	3	0	0	30	66	66	96	96	116	116	116	116
40	4	4	0	0	30	66	66	96	96	116	116	136	136
60	5	5	0	0	30	66	66	96	96	116	126	136	156

$i = 0 \text{ or } j = 0 ?$

$\rightarrow$  false

$j < w_i ?$

$\rightarrow$  false

$V(i, j) = v_i + V(i-1, j-w_i) ? \rightarrow$  false

# ແລ້ວເລືອກຂອງຫັນໄດ້ບ້າງ ?

$$V(i, j) = \begin{cases} \max(V(i-1, j), v_i + V(i-1, j-w_i)) & \text{if } i > 0 \text{ and } j \geq w_i \\ V(i-1, j) & \text{if } j < w_i \\ 0 & \text{if } i = 0 \text{ or } j = 0 \end{cases}$$

	<i>i</i>	0	1	2	3	4	5	6	7	8	9	10
<i>v<sub>i</sub></i>	<i>w<sub>i</sub></i>	0	0	0	0	0	0	0	0	0	0	0
20	2	1	0	0	20	20	20	20	20	20	20	20
30	2	2	0	0	30	30	50	50	50	50	50	50
66	3	3	0	0	30	66	66	96	96	116	116	116
40	4	4	0	0	30	66	66	96	96	116	116	136
60	5	5	0	0	30	66	66	96	96	116	126	136

 $i = 0 \text{ or } j = 0 ?$  $\rightarrow$  false $j < w_i ?$  $\rightarrow$  false $V(i, j) = v_i + V(i-1, j-w_i) ? \rightarrow$  true  $\rightarrow$  เลือกຫັນທີ 3

# แล้วเลือกของชิ้นใดบ้าง ?

$$V(i, j) = \begin{cases} \max(V(i-1, j), v_i + V(i-1, j-w_i)) & \text{if } i > 0 \text{ and } j \geq w_i \\ V(i-1, j) & \text{if } j < w_i \\ 0 & \text{if } i = 0 \text{ or } j = 0 \end{cases}$$

		j	0	1	2	3	4	5	6	7	8	9	10
i			0	1	2	3	4	5	6	7	8	9	10
$v_i$	$w_i$	0	0	0	0	0	0	0	0	0	0	0	0
20	2	1	0	0	20	20	20	20	20	20	20	20	20
30	2	2	0	0	30	30	50	50	50	50	50	50	50
66	3	3	0	0	30	66	66	96	96	116	116	116	116
40	4	4	0	0	30	66	66	96	96	116	116	136	136
60	5	5	0	0	30	66	66	96	96	116	126	136	156

$i = 0 \text{ or } j = 0 ?$

$\rightarrow$  false

$j < w_i ?$

$\rightarrow$  false

$V(i, j) = v_i + V(i-1, j-w_i) ? \rightarrow$  true  $\rightarrow$  เลือกชิ้นที่ 2

# ແລ້ວເລືອກຂອງຫັນໄດ້ບ້າງ ?

$$V(i, j) = \begin{cases} \max(V(i-1, j), v_i + V(i-1, j - w_i)) & \text{if } i > 0 \text{ and } j \geq w_i \\ V(i-1, j) & \text{if } j < w_i \\ 0 & \text{if } i = 0 \text{ or } j = 0 \end{cases}$$

	i	j	0	1	2	3	4	5	6	7	8	9	10
$v_i$	$w_i$	0	0	0	0	0	0	0	0	0	0	0	0
20	2	1	0	20	20	20	20	20	20	20	20	20	20
(30)	2	2	0	0	30	30	50	50	50	50	50	50	50
(66)	3	3	0	0	30	66	66	96	96	116	116	116	116
40	4	4	0	0	30	66	66	96	96	116	116	136	136
(60)	5	5	0	0	30	66	66	96	96	116	126	136	156

 $i = 0 \text{ or } j = 0 ?$  $\rightarrow$  true

## knapsack\_Soln(v, w, V)

```
knapsack_Soln( v[1..n], w[1..n], V[0..n][0..W] ) {  
    S = an empty set  
    i = n; j = W  
    while (i > 0 AND j > 0) {  
        if (j >= w[i] AND  
            V[i][j] == v[i] + V[i-1][j - w[i]]) {  
            S.add(i);  
            j = j - w[i];  
        }  
        i--  
    }  
    return S;  
}
```

# THONGEN

- ❖ มีเหรียญอยู่  $n$  แบบ :  $1, 2, 3, \dots, n$
- ❖ แต่ละแบบมีค่า :  $v_1, v_2, v_3, \dots, v_n$
- ❖ ต้องการทอนเงินให้ลูกค้ามูลค่า  $V$
- ❖ โดยใช้เหรียญเป็นจำนวนน้อยสุด
  - ❖ (มีเหรียญแต่ละแบบจำนวนไม่จำกัด)
- ❖ ตัวอย่าง
  - ❖ มีเหรียญ :  $1, 3, 4, 10$
  - ❖ ต้องการทอนเงินมูลค่า  $6$
  - ❖ คำตอบ :  $3, 3$

ลองทำเอง  
คล้าย ๆ Knapsack

# Sum of Subset

## ❖ กำหนดให้

- ❖ S เป็นเซตของจำนวน
- ❖ K เป็นจำนวน

ลองทำเอง  
คล้าย ๆ Knapsack

## ❖ จงหาว่า

- ❖ มีเซตย่อยของ S ที่ผลรวมของจำนวนมีค่าเท่ากับ K หรือไม่

## ❖ ตัวอย่าง

- ❖  $S = \{1, 3, 2, 9, 8\}$ ,  $K = 11 \rightarrow$  คำตอบ : มี
- ❖  $S = \{1, 3, 2, 9, 8\}$ ,  $K = 7 \rightarrow$  คำตอบ : ไม่มี

# Maximum Contiguous Sum

❖ กำหนดให้

- ❖  $A$  คือลำดับของจำนวน  $\langle a_1, a_2, \dots, a_n \rangle$

❖ จงหา

- ❖ ช่วงของข้อมูลที่ติดกันใน  $A$  ที่ผลรวมมีค่ามากสุด

❖ ตัวอย่าง

- ❖  $A = \langle 31, -41, 59, 26, -53, 58, 97, -93, -23, 84 \rangle$

❖ แนวคิด

- ❖ ลูยกุ่กช่วงของข้อมูล มีหั้งหมด  $C(n,2)$  ช่วง ใช้เวลา  $\Theta(n^2)$
- ❖ ใช้การแบ่งแยกและเอาชนะ

ลองทำเอง

- ❖ แบ่งครึ่ง, หาฝั่งซ้าย, หาฝั่งขวา, หาข้ามฝั่ง
- ❖ ใช้เวลา  $\Theta(n \log n)$

# Dynamic Programming

## ❖ เขียน recurrence ของผลรวม

- ❖  $P(i)$  คือผลรวมของข้อมูลที่ติดกันใน  $A$  ที่มีค่ามากสุด โดยให้ตัวท้ายสุดอยู่ที่ตำแหน่ง  $i$

$$P(i) = \max( P(i-1) + a_i, a_i ) \quad P(1) = a_1$$

- ❖  $S(i)$  คือผลรวมของข้อมูลที่ติดกันใน  $A[1..i]$  ที่มีค่ามากสุด (คำตอบก็คือ  $S(n)$ )

$$S(i) = \max_{1 \leq k \leq i} \{ P(k) \} = \max( \max_{1 \leq k \leq i-1} \{ P(k) \}, P(i) )$$

$$S(i) = \max( S(i-1), P(i) ) \quad S(1) = a_1$$

$$A = <2, 1, -4, 2> \rightarrow P(1) = 2, P(2) = 3, P(3) = -1, P(4) = 2 \\ S(1) = 2, S(2) = 3, S(3) = 3, S(4) = 3$$

# Dynamic Programming

```

MCS_Value(A[1..n]) {
    create S[1..n], P[1..n];
    P[1] = A[1];
    for (i = 2; i <= n; i++)
        P[i] = max(P[i - 1] + A[i], A[i]);
    S[1] = A[1];
    for (i = 2; i <= n; i++)
        S[i] = max(S[i - 1], P[i]);
    return S;
}

```

$$P(i) = \max( P(i-1) + a_i, a_i )$$

$$S(i) = \max( S(i-1), P(i) )$$

ทำอย่างไร ถ้าต้องการ  
หาช่วงของ A ที่ผลรวมมี  
ค่ามากสุด

```

MCS_Value(A[1..n]) {
    P = A[1]; S = A[1];
    for (i = 2; i <= n; i++) {
        P = max(P + A[i], A[i]);
        S = max(S, P);
    }
    return S;
}

```

 $\Theta(n)$

# ตัดข้อความเป็นคำ ๆ

- ❖ **Input** : ข้อความภาษาไทย
- ❖ **Output** : รายการสิ้นสุดของคำที่ตัดจากข้อความ

นายกนกรนดิ้ง



นาย กนก กรน ดิ้ง

นายกนกร

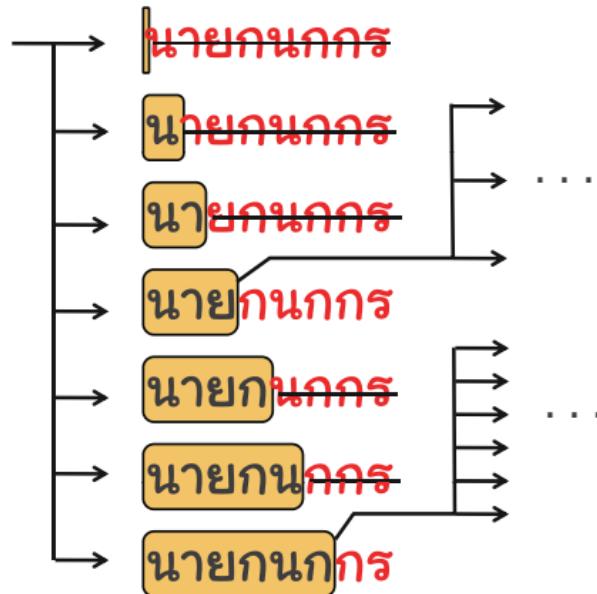


นาย กนกร

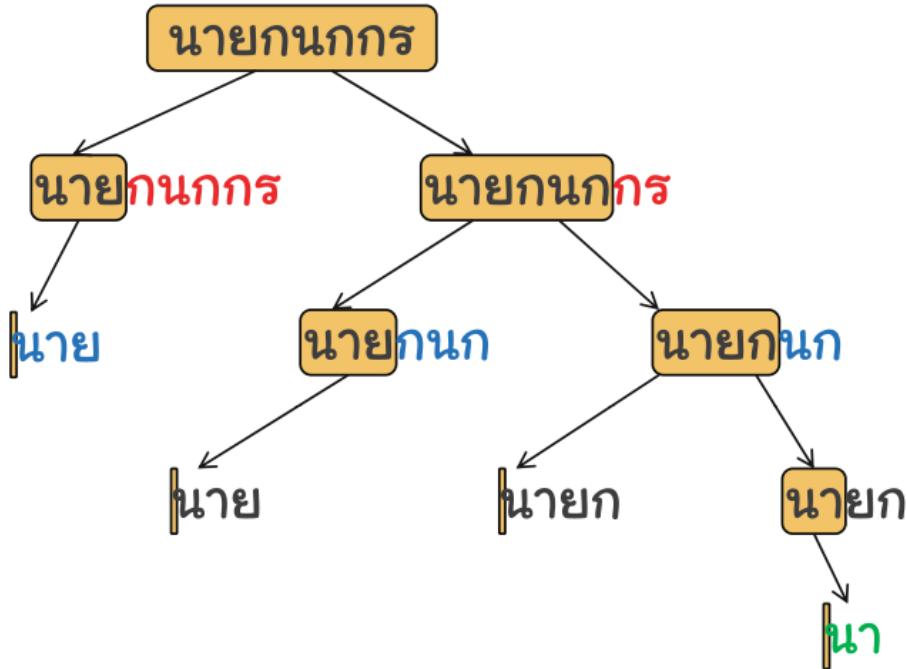
แบ่งคำโดยใช้พจนานุกรม

# ตัดข้อความเป็นคำ ๆ : Top-Down

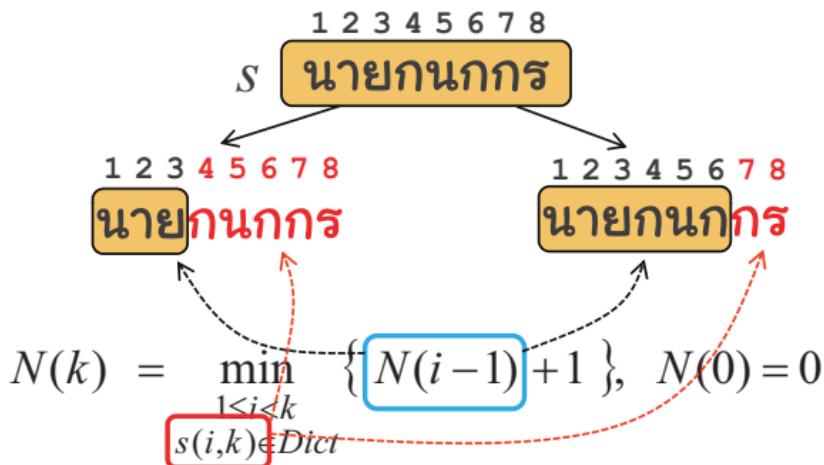
นายกนกร



# ตัดข้อความเป็นคำ ๑ : Top-Down



# ตัดข้อความเป็นคำ ๑ : Top-Down



ให้  $s$  แทนสตริงของข้อความที่ได้รับ  
 $s(i, k)$  แทนสตริงย่ออย่างตั้งแต่ตัวที่  $i$  ถึง  $k$   
 $N(k)$  แทนจำนวนคำน้อยสุดที่แบ่งได้จาก  $s(1, k)$

# ตัดข้อความเป็นคำ ๑ : Top-Down

$$N(k) = \min_{\substack{1 \leq i < k \\ s(i,k) \in \text{Dict}}} \{ N(i-1) + 1 \}, \quad N(0) = 0$$

```
N( dict, s[1..n], k ) {
    if (k == 0) return 0
    minN = ∞
    for (i = 1; i < k; i++) {
        if ( s[i..k] ∈ dict ) {
            minN = min( minN, 1 + N( dict, s, i-1 ) )
        }
    }
    return minN }
```

จำนวนค่าน้อยสุดที่  
แบ่งจากข้อความ

# ต้องการรายการของคำ

```

WordSep( dict, s[1..n], k ) {
    if (k == 0) return an empty list
    minN = ∞
    minWords = null
    for (i = 1; i < k; i++) {
        if ( s[i..k] ∈ dict ) {
            words = WordSep( dict, s, i-1 )
            if (words ≠ null) {
                words.add( s[i..k] )
                if ( words.size() < minN ) {
                    minN = words.size()
                    minWords = words
                }
            }
        }
    }
    return minWords
}

```

Overlapping Subproblems  
(ลองยกตัวอย่าง)

## ตัดข้อความเป็นคำ ๑ : Bottom-Up

$$N(k) = \min_{\substack{1 \leq i < k \\ s(i,k) \in \text{Dict}}} \{ N(i-1) + 1 \}, \quad N(0) = 0$$

$k = 2$

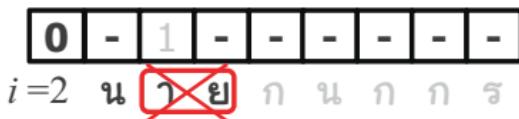
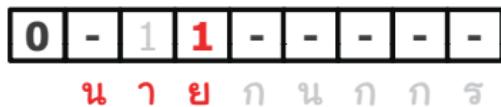
**0** - **1** - - - - -  
 $i=1$  น า ย ก น ก ก ร

**0** - **1** - - - - -  
น า ย ก น ก ก ร

# ตัดข้อความเป็นคำ 1 : Bottom-Up

$$N(k) = \min_{\substack{1 \leq i < k \\ s(i,k) \in \text{Dict}}} \{ N(i-1) + 1 \}, \quad N(0) = 0$$

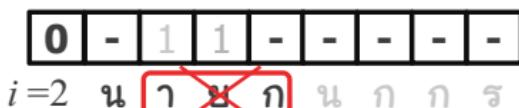
$k = 3$



# ตัดข้อความเป็นคำ 1 : Bottom-Up

$$N(k) = \min_{\substack{1 \leq i < k \\ s(i,k) \in \text{Dict}}} \{ N(i-1) + 1 \}, \quad N(0) = 0$$

$k = 4$



# ตัดข้อความเป็นคำ ๑ : Bottom-Up

$$N(k) = \min_{\substack{1 \leq i < k \\ s(i,k) \in \text{Dict}}} \{ N(i-1) + 1 \}, \quad N(0) = 0$$

$k = 5$

0	-	1	1	1	-	-	-	-
---	---	---	---	---	---	---	---	---

$i=1$  ն ա չ ղ ն ղ ղ թ

0	-	1	1	1	2	-	-	-
---	---	---	---	---	---	---	---	---

ն ա յ ղ ն ղ ղ թ

0	-	1	1	1	-	-	-	-
---	---	---	---	---	---	---	---	---

$i=2$  ն ա յ ղ ն ղ ղ թ

0	-	1	1	1	-	-	-	-
---	---	---	---	---	---	---	---	---

$i=3$  ն ա յ ղ ն ղ ղ թ

0	-	1	1	1	2	-	-	-
---	---	---	---	---	---	---	---	---

$i=4$  ն ա յ ղ ն ղ ղ թ

# ตัดข้อความเป็นคำ ๑ : Bottom-Up

$$N(k) = \min_{\substack{1 \leq i < k \\ s(i,k) \in \text{Dict}}} \{ N(i-1) + 1 \}, \quad N(0) = 0$$

$k = 6$

0	-	1	1	1	2	-	-	-
$i=1$	ນ	າ	ຍ	ກ	ນ	ກ	ກ	ຮ

0	-	1	1	1	2	2	-	-
	ນ	າ	ຍ	ກ	ນ	ກ	ກ	ຮ

0	-	1	1	1	2	-	-	-
$i=2$	ນ	າ	ຍ	ກ	ນ	ກ	ກ	ຮ

น າ ຍ ก ນ ກ ກ ລ

0	-	1	1	1	2	-	-	-
$i=3$	ນ	າ	ຍ	ກ	ນ	ກ	ກ	ຮ

ນ າ ຍ ກ ນ ກ ກ ລ

0	-	1	1	1	2	-	-	-
$i=4$	ນ	າ	ຍ	ກ	ນ	ກ	ກ	ຮ

0	-	1	1	1	2	2	-	-
$i=5$	ນ	າ	ຍ	ກ	ນ	ກ	ກ	ຮ

0	-	1	1	1	1	2	-	-
$i=6$	ນ	າ	ຍ	ກ	ນ	ກ	ກ	ຮ

0	-	1	1	1	1	2	2	-
$i=7$	ນ	າ	ຍ	ກ	ນ	ກ	ກ	ຮ

# ตัดข้อความเป็นคำ 1 : Bottom-Up

$$N(k) = \min_{\substack{1 \leq i < k \\ s(i,k) \in \text{Dict}}} \{ N(i-1) + 1 \}, \quad N(0) = 0$$

$k = 7$

0	-	1	1	1	2	2	-	-
$i=1$	ນ	າ	ຍ	ກ	ນ	ກ	ກ	ຮ

0	-	1	1	1	2	2	-	-
$i=2$	ນ	າ	ຍ	ກ	ນ	ກ	ກ	ຮ

0	-	1	1	1	2	2	-	-
$i=3$	ນ	າ	ຍ	ກ	ນ	ກ	ກ	ຮ

0	-	1	1	1	2	2	-	-
$i=4$	ນ	າ	ຍ	ກ	ນ	ກ	ກ	ຮ

0	-	1	1	1	2	2	-	-
$i=5$	ນ	າ	ຍ	ກ	ນ	ກ	ກ	ຮ

0	-	1	1	1	2	2	3	-
$i=6$	ນ	າ	ຍ	ກ	ນ	ກ	ກ	ຮ

0	-	1	1	1	2	2	3	-
	ນ	າ	ຍ	ກ	ນ	ກ	ກ	ຮ

# ตัดข้อความเป็นคำ 1 : Bottom-Up

$$N(k) = \min_{\substack{1 \leq i < k \\ s(i,k) \in \text{Dict}}} \{ N(i-1) + 1 \}, \quad N(0) = 0$$

$k = 8$

0	-	1	1	1	2	2	3	-
$i=1$	น	า	ຍ	ກ	ນ	ກ	ກ	ຮ

0	-	1	1	1	2	2	3	-
$i=2$	ն	າ	ຍ	ກ	ນ	ກ	ກ	ຮ

0	-	1	1	1	2	2	3	-
$i=3$	ն	າ	ຍ	ກ	ນ	ກ	ກ	ຮ

0	-	1	1	1	2	2	3	-
$i=4$	ն	າ	ຍ	ກ	ນ	ກ	ກ	ຮ

0	-	1	1	1	2	2	3	-
$i=5$	ն	າ	ຍ	ກ	ນ	ກ	ກ	ຮ

0	-	1	1	1	2	2	3	-
$i=6$	ն	າ	ຍ	ກ	ນ	ກ	ກ	ຮ

0	-	1	1	1	2	2	3	-
$i=7$	ն	າ	ຍ	ກ	ນ	ກ	ກ	ຮ

0	-	1	1	1	2	2	3	-
	ն	າ	ຍ	ກ	ນ	ກ	ກ	ຮ

# ตัวข้อความเป็นค่า ๑ : Bottom-Up

$$N(k) = \min_{\substack{1 \leq i < k \\ s(i,k) \in Dict}} \{ N(i-1) + 1 \}, \quad N(0) = 0$$

```

N( dict, s[1..n] ) {
    N = new array[0..n]
    N[0] = 0; N[1..n] = ∞
    for( k = 2; k <= n; k++ ) {
        for( i = 1; i < k; i++ ) {
            if ( s[i..k] ∈ dict ) {
                N[k] = min( N[k], N[i-1]+1 )
            }
        }
    }
    return N
}

```

ถ้าการค้นในพจนานุกรม  
ใช้เวลา  $\Theta(1)$

$\Theta( n^2 )$

จำนวนค่าน้อยสุดที่  
แบ่งจากข้อความ

## ต้องการรายการของคำ

$$N(k) = \min_{\substack{1 \leq i < k \\ s(i,k) \in Dict}} \{ N(i-1) + 1 \}, \quad N(0) = 0$$

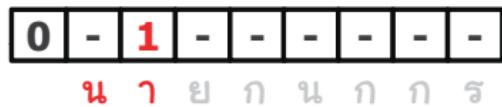
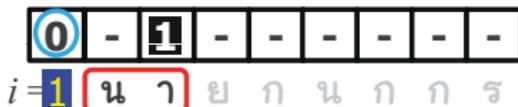
ที่แต่ละค่าของ  $k$

ต้องจำว่า  $i$  ได้ที่ได้ค่า  $N(i-1) + 1$  น้อยสุด

# จำค่า $i$ ที่ได้ min

$$N(k) = \min_{\substack{1 \leq i < k \\ s(i,k) \in \text{Dict}}} \{ N(i-1) + 1 \}, \quad N(0) = 0$$

$k = 2$



# จำนวนค่า $i$ ที่ได้ min

$$N(k) = \min_{\substack{1 \leq i < k \\ s(i,k) \in \text{Dict}}} \{ N(i-1) + 1 \}, \quad N(0) = 0$$

$k = 3$

0	-	1	1	-	-	-	-	-
$i=1$	น	า	ຍ	ก	ນ	ກ	ກ	ຮ

0	-	1	1	-	-	-	-	-
	น	າ	ຍ	ກ	ນ	ກ	ກ	ຮ

0	-	1	-	-	-	-	-	-
$i=2$	ນ	າ	ຍ	ກ	ນ	ກ	ກ	ຮ

# ຈຳຄ່າ $i$ ທີ່ໄດ້ $\min$

$$N(k) = \min_{\substack{1 \leq i < k \\ s(i,k) \in \text{Dict}}} \{ N(i-1) + 1 \}, \quad N(0) = 0$$

$k = 4$

0	-	1	1	1	-	-	-	-
$i=1$	ນ	າ	ຍ	ກ	ນ	ກ	ກ	ຮ

0	-	1	1	1	-	-	-	-
	ນ	າ	ຍ	ກ	ນ	ກ	ກ	ຮ

0	-	1	1	-	-	-	-	-
$i=2$	ນ	າ	<del>ຍ</del>	ກ	ນ	ກ	ກ	ຮ

0	-	1	1	2	-	-	-	-
$i=3$	ນ	າ	ຍ	ກ	ນ	ກ	ກ	ຮ

# จำนวน $i$ ที่ได้ min

$$N(k) = \min_{\substack{1 \leq i < k \\ s(i,k) \in \text{Dict}}} \{ N(i-1) + 1 \}, \quad N(0) = 0$$

$k = 5$

0	-	1	1	1	-	-	-	-
$i=1$	น	າ	ຢ	ກ	ນ	ກ	ກ	ຮ

0	-	1	1	1	1	2	-	-	-
	ນ	າ	ຢ	ກ	ນ	ກ	ກ	ກ	ຮ

0	-	1	1	1	-	-	-	-
$i=2$	ນ	າ	ຢ	ກ	ນ	ກ	ກ	ຮ

0	-	1	1	1	-	-	-	-
$i=3$	ນ	າ	ຢ	ກ	ນ	ກ	ກ	ຮ

0	-	1	1	1	2	-	-	-
$i=4$	ນ	າ	ຢ	ກ	ນ	ກ	ກ	ຮ

0	-	1	1	1	2	-	-	-
$i=5$	ນ	າ	ຢ	ກ	ນ	ກ	ກ	ຮ

# จำนวน i ที่ได้ min

$$N(k) = \min_{\substack{1 \leq i < k \\ s(i,k) \in \text{Dict}}} \{ N(i-1) + 1 \}, \quad N(0) = 0$$

$k = 6$

0	-	1	1	1	2	-	-	-
$i=1$	น	ว	ย	ๆ	น	ก	ก	ร

0	-	1	1	1	1	2	2	-	-
	น	ว	ย	ก	น	ก	น	ก	ร

0	-	1	1	1	2	-	-	-
$i=2$	น	ว	ย	ๆ	น	ก	ก	ร

0	-	1	1	1	2	-	-	-
$i=3$	น	ว	ย	ก	น	ก	ก	ร

0	-	1	1	1	2	2	-	-
$i=4$	น	ว	ย	ก	น	ก	ก	ร

0	-	1	1	1	2	2	-	-
$i=5$	น	ว	ย	ก	น	ก	ก	ร

# จำนวน $i$ ที่ได้ min

$$N(k) = \min_{\substack{1 \leq i < k \\ s(i,k) \in \text{Dict}}} \{ N(i-1) + 1 \}, \quad N(0) = 0$$

$k = 7$

0	-	1	1	1	2	2	-	-
$i=1$	น	า	ຍ	<del>น</del>	ກ	ກ	ຮ	

0	-	1	1	1	2	2	-	-
$i=2$	ນ	າ	ຍ	<del>ກ</del>	ນ	ກ	ຮ	

0	-	1	1	1	2	2	-	-
$i=3$	ນ	າ	ຍ	<del>ກ</del>	ນ	ກ	ຮ	

0	-	1	1	1	2	2	-	-
$i=4$	ນ	າ	ຍ	<del>ກ</del>	ນ	<del>ກ</del>	ຮ	

0	-	1	1	1	2	2	-	-
$i=5$	ນ	າ	ຍ	ກ	<del>ນ</del>	<del>ກ</del>	ຮ	

0	-	1	1	1	2	2	3	-
$i=6$	ນ	າ	ຍ	ກ	ນ	<del>ກ</del>	<del>ກ</del>	ຮ

0	-	1	1	1	1	2	2	3	-
	ນ	າ	ຍ	ກ	ນ	<del>ກ</del>	<del>ກ</del>	ຮ	

# จำค่า i ที่ได้ min

$$N(k) = \min_{\substack{1 \leq i < k \\ s(i,k) \in \text{Dict}}} \{ N(i-1) + 1 \}, \quad N(0) = 0$$

$k = 8$

0	-	1	1	1	2	2	3	-
$i=1$	น	า	ย	ก	น	ก	ก	ร

0	-	1	1	1	2	2	3	-
$i=2$	น	า	ย	ก	%	น	ก	ก

0	-	1	1	1	2	2	3	-
$i=3$	น	า	ย	ก	%	ก	ก	ร

0	-	1	1	1	2	2	3	2
$i=4$	น	า	ย	ก	%	ก	ก	ร

0	-	1	1	1	2	2	3	-
$i=5$	น	า	ย	ก	%	ก	ก	ร

0	-	1	1	1	2	2	3	-
$i=6$	น	า	ย	ก	%	ก	ก	ร

0	-	1	1	1	2	2	3	3
$i=7$	น	า	ย	ก	%	ก	ก	ร

0	-	1	1	1	2	2	3	2
		1	1	1	4	4	6	

0	-	1	1	1	2	2	3	2
		น	า	ย	ก	%	ก	ร

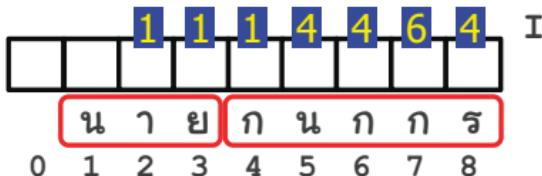
# ตัดข้อความเป็นคำ ๑ : Bottom-Up

$$N(k) = \min_{\substack{1 \leq i < k \\ s(i,k) \in \text{Dict}}} \{ N(i-1) + 1 \}, \quad N(0) = 0$$

```
WordSep( dict, s[1..n] ) {
    N = new array[0..n]
    N[0] = 0; N[1..n] = ∞
    I = new array[1..n]
    for( k = 2; k <= n; k++ )
        for( i = 1; i < k; i++ )
            if ( s[i..k] ∈ dict )
                if ( N[i-1]+1 < N[k] ) {
                    N[k] = N[i-1]+1
                    I[k] = i
                }
    }
}
```

0	-	1	1	1	1	2	2	2	3	2	I
น	า	ย	ก	น	ก	ก	ร	ส	ก	ร	s

# ตัดข้อความเป็นคำ 1 : Bottom-Up



```

minWords = an empty list
k = n
while (k > 0) {
    minWords = s[ I[k] , k ] + minWords
    k = I[k] - 1
}
return minWords
}

```

# Longest Increasing Subsequence

- ❖ กำหนดให้

- ❖  $A$  คือลำดับของจำนวน  $\langle a_1, a_2, \dots, a_n \rangle$

- ❖ จงหา

- ❖ subsequence ที่ยาวสุดของ  $A$  ที่มีค่าเรียงจากน้อยไปมาก

- ❖ ตัวอย่าง

- ❖  $A = \langle 31, -41, 59, 26, -53, 58, 97, -93, -23, 84 \rangle$

- ❖ คำตอบคือ  $\langle -41, 26, 58, 97 \rangle$

ลองทำดู

# Matrix-Chain Multiplication

$A_1 A_2 A_3 A_4$

$$\begin{aligned} & (A_1(A_2(A_3A_4))) \\ & (A_1((A_2A_3)A_4)) \\ & ((A_1A_2)(A_3A_4)) \\ & (((A_1A_2)A_3)A_4) \\ & ((A_1(A_2A_3))A_4) \end{aligned}$$

วงเล็บกำหนดลำดับการคูณ  
ลำดับการคูณต่างกันใช้  
เวลาคูณต่างกัน  
จงหาวิธีการใส่วงเล็บ  
ที่ใช้เวลาการคูณเร็วสุด

# Matrix Multiplication

$$C = A \times B \quad \begin{array}{l} \text{ต้องคูณด้วย *} \\ [p \times r] \quad [p \times q] \quad [q \times r] \quad \text{จำนวน } pqr \text{ ครั้ง} \end{array}$$

```
mult(A[1..p][1..q], B[1..q][1..r]) {  
    create C[1..p][1..r]  
    for (i = 1; i <= p; i++) {  
        for (j = 1; j <= r; j++) {  
            C[i][j] = 0  
            for (k = 1; k <= q; k++) {  
                C[i][j] += A[i][k] * B[k][j];  
            }  
        }  
    }  
    return C;  
}
```

$$c_{i,j} = \sum_{k=1}^q a_{i,k} b_{k,j}$$

# ลำดับการคูณต่างกัน ใช้เวลาต่างกัน

- ❖  $A_1 \quad A_2 \quad A_3$   
 $[10 \times 100], [100 \times 5], [5 \times 50]$
- ❖ คูณตามลำดับ  $((A_1 A_2) A_3)$ 
  - ❖  $(A_1 A_2)$  ต้องคูณด้วย \* จำนวน  $10 \times 100 \times 5 = 5,000$  ครั้ง
  - ❖ ได้เมทริกซ์  $A_{12}$  ขนาด  $[10 \times 5]$
  - ❖  $(A_{12} A_3)$  ต้องคูณด้วย \* จำนวน  $10 \times 5 \times 50 = 2,500$  ครั้ง
  - ❖ รวมเป็น  $= \underline{7,500}$  ครั้ง
- ❖ คูณตามลำดับ  $(A_1 (A_2 A_3))$ 
  - ❖  $(A_2 A_3)$  ต้องคูณด้วย \* จำนวน  $100 \times 5 \times 50 = 25,000$  ครั้ง
  - ❖ ได้เมทริกซ์  $A_{23}$  ขนาด  $[100 \times 50]$
  - ❖  $(A_1 A_{23})$  ต้องคูณด้วย \* จำนวน  $10 \times 100 \times 50 = 50,000$  ครั้ง
  - ❖ รวมเป็น  $= \underline{75,000}$  ครั้ง

# ใส่่วงเล็บได้กี่รูปแบบ

$A_1 \quad A_2 \quad A_3 \quad A_4 \quad A_5$

$(A_1) \quad (A_2 \quad A_3 \quad A_4 \quad A_5) \rightarrow 1 \times 5$

$(A_1 \quad A_2) \quad (A_3 \quad A_4 \quad A_5) \rightarrow 1 \times 2$

$(A_1 \quad A_2 \quad A_3) \quad (A_4 \quad A_5) \rightarrow 2 \times 1$

$(A_1 \quad A_2 \quad A_3 \quad A_4) \quad (A_5) \rightarrow 5 \times 1$

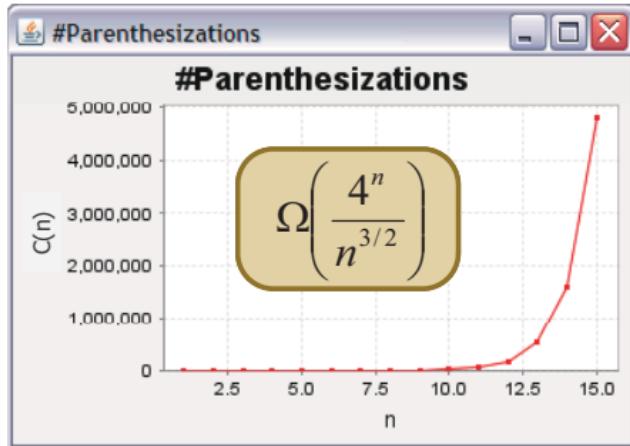
$C(n)$  คือจำนวนการใส่่วงเล็บกรณีมี  $n$  ตัว

$$\underbrace{(A_1 \dots A_k)}_{C(k)} \quad \underbrace{(A_{k+1} \dots A_n)}_{C(n-k)}$$

$$C(n) = \sum_{k=1}^{n-1} (C(k) C(n-k))$$

# ใส่วงเล็บได้กี่รูปแบบ

$$C(n) = \begin{cases} \sum_{k=1}^{n-1} (C(k)C(n-k)) & n \geq 3 \\ 1 & n \leq 2 \end{cases}$$



ข้ามมาก ๆ ถ้าต้องลองทุกรูปแบบ

## นิยาม

- ❖ ต้องการหาวิธีส่วงเล็บของ  $A_1 A_2 \dots A_n$
- ❖  $A_1$  มีขนาด  $p_0 \times p_1$
- ❖  $A_2$  มีขนาด  $p_1 \times p_2$
- ...
- ❖  $A_n$  มีขนาด  $p_{n-1} \times p_n$
- ❖  $A_i \dots A_j$  มีขนาด  $p_{i-1} \times p_j$
- ❖ ยังไม่หาริธีการส่วงเล็บที่ดีสุด
- ❖ ขอหาจำนวนการคูณด้วย \* ของการส่วงเล็บที่ดีสุด
- ❖  $m(i, j) =$ จำนวน \* น้อยสุดเพื่อหาผลคูณ  $A_i \dots A_j$
- ❖  $m(1, n)$  คือคำตอบที่ต้องการ

# จำนวนการคูณ

$m(i, j)$  คือจำนวนการคูณด้วย \* น้อยสุดของการคูณ  $\mathbf{A}_i \mathbf{A}_{i+1} \dots \mathbf{A}_j$

คูณ  $\mathbf{A}_i \dots \mathbf{A}_k$  ใช้  $m(i, k)$  ได้เมทริกซ์ขนาด  $[p_{i-1} \times p_k]$

คูณ  $\mathbf{A}_{k+1} \dots \mathbf{A}_j$  ใช้  $m(k+1, j)$  ได้เมทริกซ์ขนาด  $[p_k \times p_j]$

คูณเมทริกซ์ทั้งสองใช้  $p_{i-1} p_k p_j$

$$\underbrace{(\mathbf{A}_i \dots \mathbf{A}_k)}_{[p_{i-1} \times p_k]} \quad \underbrace{(\mathbf{A}_{k+1} \dots \mathbf{A}_j)}_{[p_k \times p_j]}$$

$$m(i, k) + m(k+1, j) + p_{i-1} p_k p_j$$

จำนวนการคูณเดียว \* น้อยสุด เมื่อแบ่งที่  $k$

# จำนวนการคูณน้อยสุด

$m(i, j)$  คือจำนวนการคูณด้วย \* น้อยสุดของการคูณ  $\mathbf{A}_i \mathbf{A}_{i+1} \dots \mathbf{A}_j$

$$\underbrace{(\mathbf{A}_i \dots \mathbf{A}_k)}_{m(i, k)} + \underbrace{(\mathbf{A}_{k+1} \dots \mathbf{A}_j)}_{m(k+1, j)} + p_{i-1} p_k p_j$$

$$(\mathbf{A}_1) (\mathbf{A}_2 \quad \mathbf{A}_3 \quad \mathbf{A}_4 \quad \mathbf{A}_5) \quad m(1,1) + m(2,5) + p_0 p_1 p_5$$

$$(\mathbf{A}_1 \quad \mathbf{A}_2) (\mathbf{A}_3 \quad \mathbf{A}_4 \quad \mathbf{A}_5) \quad m(1,2) + m(3,5) + p_0 p_2 p_5$$

$$(\mathbf{A}_1 \quad \mathbf{A}_2 \quad \mathbf{A}_3) (\mathbf{A}_4 \quad \mathbf{A}_5) \quad m(1,3) + m(4,5) + p_0 p_3 p_5$$

$$(\mathbf{A}_1 \quad \mathbf{A}_2 \quad \mathbf{A}_3 \quad \mathbf{A}_4) (\mathbf{A}_5) \quad m(1,4) + m(5,5) + p_0 p_4 p_5$$

$$m(i, j) = \begin{cases} \min_{i \leq k \leq j-1} \left\{ m(i, k) + m(k+1, j) + p_{i-1} p_k p_j \right\} & \text{if } i < j \\ 0 & \text{if } i = j \end{cases}$$

# MCM : Top-down

$$m(i, j) = \begin{cases} \min_{i \leq k \leq j-1} \left\{ m(i, k) + m(k+1, j) + p_{i-1} p_k p_j \right\} & \text{if } i < j \\ 0 & \text{if } i = j \end{cases}$$

```
mcm( p[0..n], i, j ) {
    if (i == j) return 0;

    minMCM = ∞;
    for( k = i; k <= j-1; k++ ) {
        minMCM = min( minMCM,
                        mcm(p, i, k) +
                        mcm(p, k+1, j) +
                        p[i-1]*p[k]*p[j] );
    }
    return minMCM;
}
```

# Top-down + Memoization

$$m(i, j) = \begin{cases} \min_{i \leq k \leq j-1} \left\{ m(i, k) + m(k+1, j) + p_{i-1} p_k p_j \right\} & \text{if } i < j \\ 0 & \text{if } i = j \end{cases}$$

```
mcm( p[0..n], i, j , M[1..n][1..n] ) {  
    if (i == j) return 0;  
    if (M[i][j] > 0) return M[i][j]  
    M[i][j] = ∞  
    for( k = i; k <= j-1; k++ ) {  
        M[i][j]= min( M[i][j] ,  
                        mcm(p, i, k, M) +  
                        mcm(p, k+1, j, M) +  
                        p[i-1]*p[k]*p[j] );  
    }  
    return M[i][j]  
}
```

# MCM : Bottom-Up

$$m(i, j) = \begin{cases} \min_{i \leq k \leq j-1} \left\{ m(i, k) + m(k+1, j) + p_{i-1} p_k p_j \right\} & \text{if } i < j \\ 0 & \text{if } i = j \end{cases}$$

j \ i	1	2	3	4	5
1					
2		m(2,2)	m(2,3)	m(2,4)	m(2,5)
3					m(3,5)
4					m(4,5)
5					m(5,5)

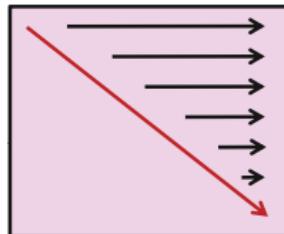
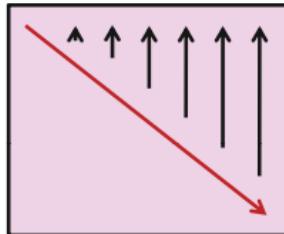
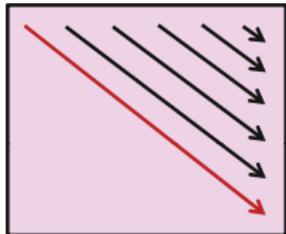
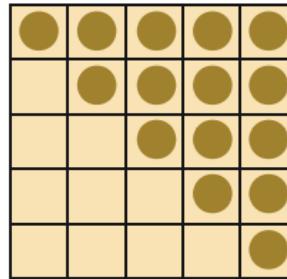
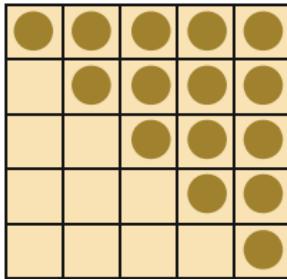
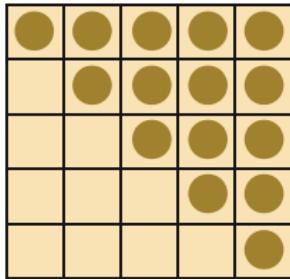
# คำตอบใหญ่ได้จากคำตอบย่อย

$$m(i, j) = \begin{cases} \min_{i \leq k \leq j-1} \left\{ m(i, k) + m(k+1, j) + p_{i-1} p_k p_j \right\} & \text{if } i < j \\ 0 & \text{if } i = j \end{cases}$$

i \ j	1	2	3	4	5
1	$m(1,1)$	$m(2,2)$	$m(2,3)$	$m(2,4)$	$m(1,5)$
2					$m(2,5)$
3					$m(3,5)$
4					$m(4,5)$
5					$m(5,5)$

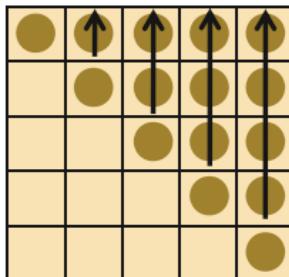
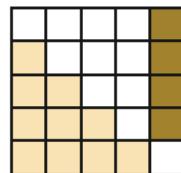
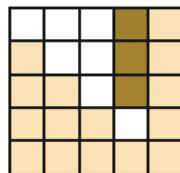
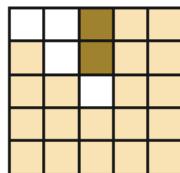
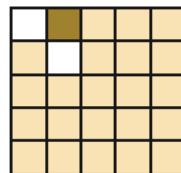
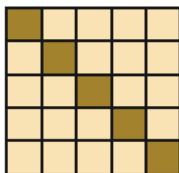
# เติมตารางได้หลายรูปแบบ

เติมคำตอบเล็กก่อนคำตอบใหญ่



# รูปแบบการเติมตารางคำตอบ

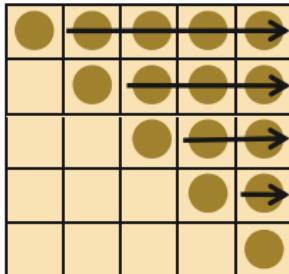
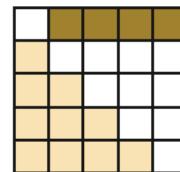
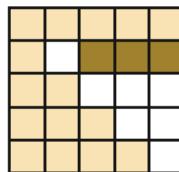
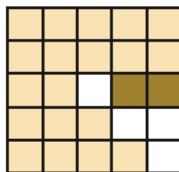
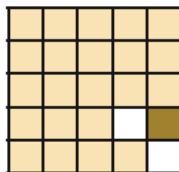
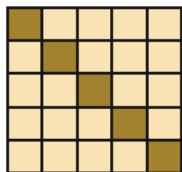
$$m(i, j) = \begin{cases} \min_{i \leq k \leq j-1} \left\{ m(i, k) + m(k+1, j) + p_{i-1} p_k p_j \right\} & \text{if } i < j \\ 0 & \text{if } i = j \end{cases}$$



```
for( i = 1; i <= n; i++ )  
    m[i][i] = 0  
for( j = 2; j <= n; j++ ) {  
    for( i = j-1; i >= 1; i-- ) {  
        m[i][j] = ...  
    }  
}
```

# รูปแบบการเติมตารางคำตอบ

$$m(i, j) = \begin{cases} \min_{i \leq k \leq j-1} \left\{ m(i, k) + m(k+1, j) + p_{i-1} p_k p_j \right\} & \text{if } i < j \\ 0 & \text{if } i = j \end{cases}$$



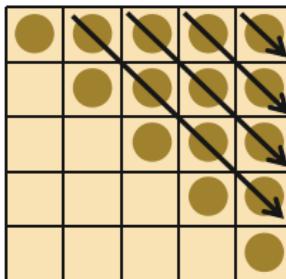
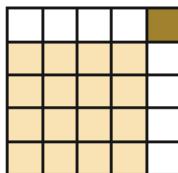
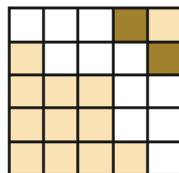
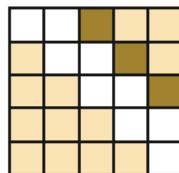
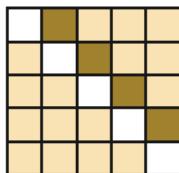
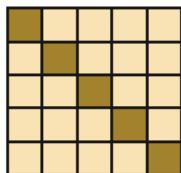
```

for( i = 1; i <= n; i++ )
    m[i][i] = 0
for( i = n-1; i >= 1; i-- ) {
    for ( j = i+1; j <= n; j++ ) {
        m[i][j] = ...
    }
}

```

# รูปแบบการเติมตารางคำตอบ

$$m(i, j) = \begin{cases} \min_{i \leq k \leq j-1} \{ m(i, k) + m(k+1, j) + p_{i-1} p_k p_j \} & \text{if } i < j \\ 0 & \text{if } i = j \end{cases}$$



# ตัวอย่าง

$$m(i, j) = \begin{cases} \min_{i \leq k \leq j-1} \left\{ m(i, k) + m(k+1, j) + p_{i-1} p_k p_j \right\} & \text{if } i < j \\ 0 & \text{if } i = j \end{cases}$$

$A_1 A_2 A_3 A_4 A_5$

$10 \times 5 \times 1 \times 5 \times 10 \times 2$

	$A_1$	$A_2$	$A_3$	$A_4$	$A_5$
$i$	1	2	3	4	5
$j$					
1	(0)	50			
2		(0)	25		
3			(0)	50	
4				(0)	100
5					(0)

# ตัวอย่าง

$$m(i, j) = \begin{cases} \min_{i \leq k \leq j-1} \left\{ m(i, k) + m(k+1, j) + p_{i-1} p_k p_j \right\} & \text{if } i < j \\ 0 & \text{if } i = j \end{cases}$$

$\mathbf{A_1 A_2 A_3 A_4 A_5}$   
**10**  $\times$  **5**  $\times$  **10**  $\times$  **2**

		$A_1$	$A_2$	$A_3$	$A_4$	$A_5$
		1	2	3	4	5
$i$	$j$	1	0	50	100	
		2		0	25	
3				0	50	
4					0	100
5						0

$$0 + 25 + 10 \times 5 \times 5 = 275$$

$$50 + 0 + 10 \times 1 \times 5 = 100$$

# ตัวอย่าง

$$m(i, j) = \begin{cases} \min_{i \leq k \leq j-1} \{ m(i, k) + m(k+1, j) + p_{i-1} p_k p_j \} & \text{if } i < j \\ 0 & \text{if } i = j \end{cases}$$

$10 \times 5 \times [ ] \times 10 \times 2$

		$A_1$	$A_2$	$A_3$	$A_4$	$A_5$
		1	2	3	4	5
i	j	0	50	100		
1	2		0	25	100	
2	3			0	50	
3	4				0	100
4	5					0

$$0 + 50 + 5 \times 1 \times 10 = 100$$

$$25 + 0 + 5 \times 5 \times 10 = 275$$

# ตัวอย่าง

$$m(i, j) = \begin{cases} \min_{i \leq k \leq j-1} \{ m(i, k) + m(k+1, j) + p_{i-1} p_k p_j \} & \text{if } i < j \\ 0 & \text{if } i = j \end{cases}$$

$10 \times 5 \times 1 \times \boxed{\dots} \times 2$

		$A_1$	$A_2$	$A_3$	$A_4$	$A_5$
		1	2	3	4	5
i \ j	1	0	50	100		
2		0	25	100		
3			0	50	70	
4				0	100	
5					0	

$$0 + 100 + 1 \times 5 \times 2 = 110$$

$$50 + 0 + 1 \times 10 \times 2 = \boxed{70}$$

# ตัวอย่าง

$$m(i, j) = \begin{cases} \min_{i \leq k \leq j-1} \{ m(i, k) + m(k+1, j) + p_{i-1} p_k p_j \} & \text{if } i < j \\ 0 & \text{if } i = j \end{cases}$$

$10 \times$  [ ]  $10 \times 2$

	$A_1$	$A_2$	$A_3$	$A_4$	$A_5$
i	1	2	3	4	5
j	1	2	3	4	5
1	0	50	100	200	
2		0	25	100	
3			0	50	70
4				0	100
5					0

$$0 + 100 + 10 \times 5 \times 10 = 600$$

$$50 + 50 + 10 \times 1 \times 10 = 200$$

$$100 + 0 + 10 \times 5 \times 10 = 600$$

# ตัวอย่าง

$$m(i, j) = \begin{cases} \min_{i \leq k \leq j-1} \left\{ m(i, k) + m(k+1, j) + p_{i-1} p_k p_j \right\} & \text{if } i < j \\ 0 & \text{if } i = j \end{cases}$$

$$10 \times 5 \times \boxed{2}$$

	$A_1$	$A_2$	$A_3$	$A_4$	$A_5$
$i$	1	2	3	4	5
$j$					
1	0	50	100	200	
2		0	25	100	80
3			0	50	70
4				0	100
5					0

$$0 + 70 + 5 \times 1 \times 2 = 80$$

$$25 + 100 + 5 \times 5 \times 2 = 175$$

$$100 + 0 + 5 \times 10 \times 2 = 200$$

# ตัวอย่าง

$$m(i, j) = \begin{cases} \min_{i \leq k \leq j-1} \{ m(i, k) + m(k+1, j) + p_{i-1} p_k p_j \} & \text{if } i < j \\ 0 & \text{if } i = j \end{cases}$$

$10 \times$   2

	A <sub>1</sub>	A <sub>2</sub>	A <sub>3</sub>	A <sub>4</sub>	A <sub>5</sub>
i \ j	1	2	3	4	5
1	0	50	100	200	140
2		0	25	100	80
3			0	50	70
4				0	100
5					0

$0 + 80 + 10 \times 5 \times 2 = 180$

$50 + 70 + 10 \times 1 \times 2 = 140$

$100 + 100 + 10 \times 5 \times 2 = 300$

$200 + 0 + 10 \times 10 \times 2 = 400$

# วิธีการเติมตารางคำตอบ

$$m(i, j) = \begin{cases} \min_{i \leq k \leq j-1} \{ m(i, k) + m(k+1, j) + p_{i-1} p_k p_j \} & \text{if } i < j \\ 0 & \text{if } i = j \end{cases}$$

```

for( i = 1; i <= n; i++) M[i][i]=0
for( d = 1; d < n; d++ ) {
    for ( i = 1; i <= n-d; i++ ) {
        j = i + d;
        for ( k = i; k < j; k++ ) {
            M[i][j] = ...
        }
    }
}

```

d = 1	d = 2	d = 3	d = 4
i, j	i, j	i, j	i, j
1, 2	1, 3	1, 4	1, 5
2, 3	2, 4	2, 5	
3, 4	3, 5		
4, 5			

1,1	1,2	1,3	1,4	1,5
2,2	2,3	2,4	2,5	
3,3	3,4	3,5		
4,4	4,5			
5,5				

# MCM : Bottom-up

$$m(i, j) = \begin{cases} \min_{i \leq k \leq j-1} \left\{ m(i, k) + m(k+1, j) + p_{i-1} p_k p_j \right\} & \text{if } i < j \\ 0 & \text{if } i = j \end{cases}$$

```
mcm_Value( p[0..n] ) {
    M = new array[1..n][1..n]
    for( i = 1; i <= n; i++ ) M[i][i] = 0
    for( d = 1; d < n; d++ ) {
        for ( i = 1; i <= n-d; i++ ) {
            j = i + d;
            M[i][j] = infinity
            for ( k = i; k < j; k++ ) {
                M[i][j] = min( M[i][j],
                    M[i][k] + M[k+1][j] + p[i-1]*p[k]*p[j]);
            }
        }
    }
    return M;
}
```

$\Theta(n^3)$

# จำจุดแบ่ง k ที่ได้ค่าน้อยสุด

i \ j	1	2	3	4	5
1	0	1 50	1 100	2 200	2 140
2		0	2 25	2 100	2 80
3			0	3 50	4 70
4				0	4 100
5					0

( $\mathbf{A}_i \dots \mathbf{A}_k$ ) ( $\mathbf{A}_{k+1} \dots \mathbf{A}_j$ )

$$m(i, j) = \begin{cases} \min_{i \leq k \leq j-1} \{ m(i, k) + m(k+1, j) + p_{i-1} p_k p_j \} & \text{if } i < j \\ 0 & \text{if } i = j \end{cases}$$

# การใส่ร่องเล็บที่คูณเร็วสุด

i \ j	1	2	3	4	5
1	0	1 50	1 100	2 200	2 140
2		0	2 25	2 100	2 80
3			0	3 50	4 70
4				0	4 100
5					0

- $[1, 5] \rightarrow 2 \quad (A_1 \ A_2) (A_3 \ A_4 \ A_5)$   
 $[1, 2] \rightarrow 1 \quad ((A_1) (A_2)) (A_3 \ A_4 \ A_5)$   
 $[3, 5] \rightarrow 4 \quad ((A_1) (A_2)) ((A_3 \ A_4) (A_5))$   
 $[3, 4] \rightarrow 3 \quad ((A_1) (A_2)) (((A_3) (A_4)) (A_5))$

# ຈຳຈຸດແບ່ງ matrix chain

```
mcm_Order ( p[0..n] ) {  
    M = new Array[1..n][1..n]  
    K = new Array[1..n][1..n]  
    for( d = 1; d < n; d++ ) {  
        for ( i = 1; i <= n-d; i++ ) {  
            j = i + d;  
            M[i][j] = ∞  
            for ( k = i; k < j; k++ ) {  
                t = M[i][k] + M[k+1][j] + p[i-1]*p[k]*p[j];  
                if (t < M[i][j]) {  
                    M[i][j] = t;  
                    K[i][j] = k;  
                }  
            }  
        }  
    }  
    return K;  
}
```

$(A_i \dots A_k) (A_{k+1} \dots A_j)$

# MCM : คูณเลย

```
mcm_Mult( A[1..n], p[0..n] ) {
    K = mcm_Order( p );
    return mcm_Mult( A, K, 1, n );
}
```

 $\Theta(n^3)$  $\Theta(?)$ 

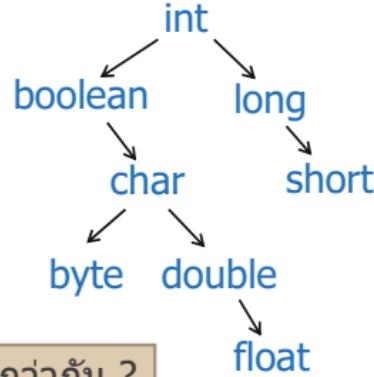
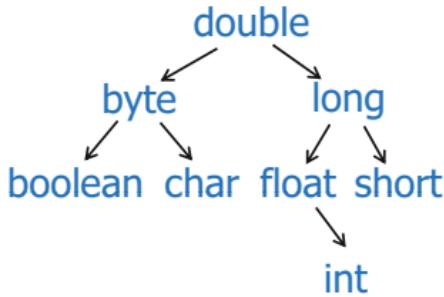
```
mcm_Mult( A[1..n], K[1..n][1..n], i, j ) {
    if (i == j) return A[i];
    X = mcm_Mult( A, K, i, K[i][j] );
    Y = mcm_Mult( A, K, K[i][j]+1, j );
    return mult(X, Y);
}
```

$$(A_i \dots A_k) (A_{k+1} \dots A_j)$$

$$X \quad \times \quad Y$$

# Binary Search Tree

- ❖ ต้องการสร้าง binary search tree เพื่อเก็บ keywords ที่เป็น primitive data types ของจาวา
  - ❖ boolean, byte, char, double, float, int, long, short



ต้นไหนให้บริการค้นข้อมูลได้ดีกว่ากัน ?

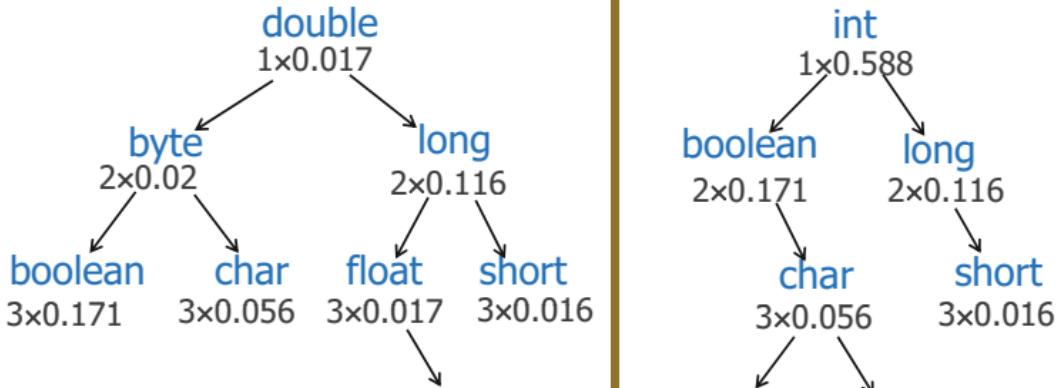
# Binary Search Tree

- ❖ เตี้ย ≠ ดี เสมอไป : ถ้าพิจารณาความถี่ของคำ
- ❖ ลองนับ primitive data types ที่ใช้ใน source codes ของ `java.util.* (jdk1.6.0_14)` พบว่า

❖ <code>int</code>	ประเภท	2,274	ครั้ง	58.76%
❖ <code>boolean</code>	"	661	"	17.08%
❖ <code>long</code>	"	450	"	11.63%
❖ <code>char</code>	"	217	"	5.61%
❖ <code>byte</code>	"	78	"	2.02%
❖ <code>double</code>	"	66	"	1.71%
❖ <code>float</code>	"	64	"	1.65%
❖ <code>short</code>	"	60	"	1.55%

# Binary Search Tree

int	boolean	long	char	byte	double	float	short
58.8%	17.1%	11.6%	5.6%	2.0%	1.7%	1.7%	1.6%



$$\sum_{k=1}^n p_k d_k$$

3.42

$p_k$  = ความน่าจะเป็นในการค้นคำที่  $k$   
 $d_k$  = จำนวนปมที่อ่านในการค้นคำที่  $k$

1.61

# Optimal Binary Search Tree

## ❖ input

- ❖ เชตของคำ และความถี่ในการใช้งานของแต่ละคำ

## ❖ output

- ❖ binary search tree ที่มี  $\sum_{k=1}^n p_k d_k$  น้อยสุด

ลองทำเอง  
คล้าย ๆ MCM



# อัลกอริทึมแบบละเอียด

สมชาย ประสิทธิ์ยุตระกูต

## หัวข้อ

- ❖ แนวคิด และโครงร่างของอัลกอริทึม
- ❖ ลักษณะของปัญหา
  - ❖ optimal substructure
  - ❖ greedy choice
- ❖ ตัวอย่างปัญหาที่ได้คำตอบ
  - ❖ ทีดีสุด
  - ❖ ทีดีเพียงพอ

# ไปทางไหนดี ?



Greedy Choice

ไปทางที่รถไม่ติด

# ทอนเงิน

- ❖ มีเหรียญอยู่  $n$  แบบ : 1, 2, 3, ...,  $n$
- ❖ แต่ละแบบมีค่า :  $v_1, v_2, v_3, \dots, v_n$
- ❖ ต้องการทอนเงินให้ลูกค้ามูลค่า  $V$
- ❖ โดยใช้เหรียญเป็นจำนวนน้อยสุด
  - ❖ (มีเหรียญแต่ละแบบจำนวนไม่จำกัด)
- ❖ ด้วยปั่ง : สกุลเงินไทย
  - ❖ มีเหรียญ : 1, 2, 5, 10 บาท (ขอไม่ใช้เหรียญสลึง)
  - ❖ ต้องการทอนเงินมูลค่า 38 บาท
  - ❖ คำตอบ : 10 บาท 3 เหรียญ,  
5 บาท 1 เหรียญ,  
2 บาท 1 เหรียญ,  
1 บาท 1 เหรียญ



Greedy Choice  
ทอนด้วยเหรียญที่มี  
ค่ามากสุดก่อน

# thon เงิน : greedy อาจใช้ไม่ได้

## ❖ สมมติว่ามีเหรียญ 4 บาท

❖ มีเหรียญ : 1, 2, 4, 5, 10 บาท

❖ ต้องการทอนเงินมูลค่า 38 บาท

❖ ทอนด้วยเหรียญที่มีค่ามากสุดก่อน

10 บาท 3 เหรียญ,

5 บาท 1 เหรียญ,

2 บาท 1 เหรียญ,

1 บาท 1 เหรียญ

ไม่ได้จำนวน  
เหรียญน้อยสุด

❖ ทอนแบบใช้จำนวนเหรียญน้อยสุด

10 บาท 3 เหรียญ,

4 บาท 2 เหรียญ,

# อัลกอริทึมแบบลักษณะ

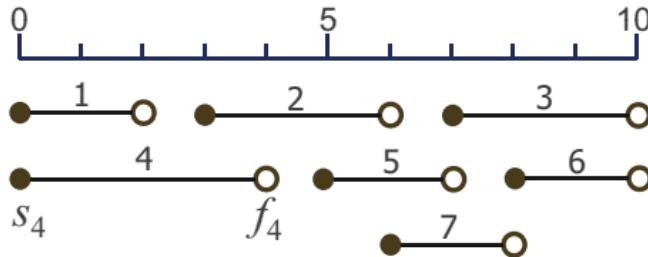
- ❖ เมื่อกับ optimization problems  
(คล้าย dynamic programming)
- ❖ เป็นวิธีของการตัดสินใจเลือกเพื่อสร้างคำตอบ
- ❖ พิจารณาทางเลือกที่ดีที่สุด ณ ปัจจุบัน  
(locally optimal choice, greedy choice)
- ❖ เพื่อสร้างคำตอบเล็ก → ในที่ → สมบูรณ์
- ❖ หวังว่าจะได้คำตอบสมบูรณ์ที่ดีที่สุด → ต้องพิสูจน์  
(globally optimal solution)

# โครงสร้างของอัลกอริทึม

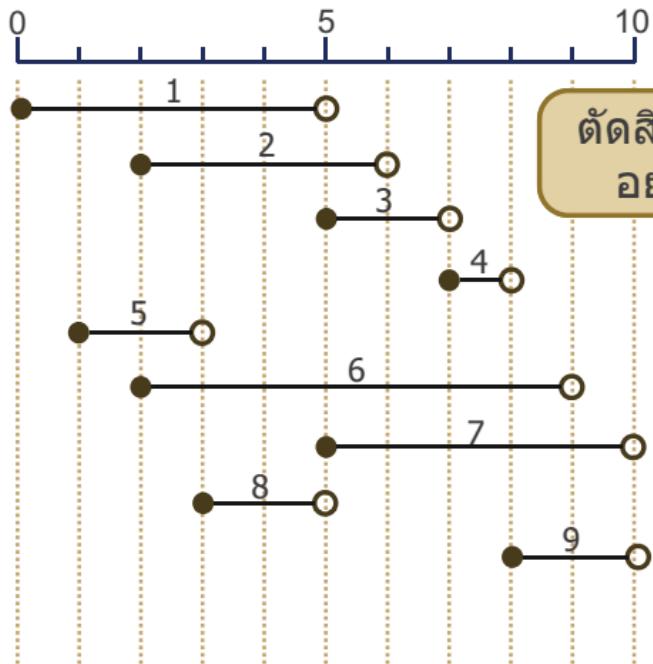
```
Greedy( C ) {  
    S = an empty set  
    while( C ≠ Ø ) {  
        x = select( C )  
        C = C - {x}  
        if ( isFeasible( S ∪ {x} ) ) {  
            S = S ∪ {x}  
            if ( isSolution( S ) ) return S  
        }  
    }  
    return "No Solution"  
}
```

# Activity Selection

- ❖ **Input :** มีงาน  $n$  งานที่ต้องใช้ห้องจัดงานห้องหนึ่ง
  - $s_i$  แทนเวลา เริ่มใช้ห้องของงาน  $i$
  - $f_i$  แทนเวลา เลิกใช้ห้องของงาน  $i$
- ❖ **Output:** กลุ่มของงานจำนวนมากสุดที่ไม่ใช้ห้องในเวลาเดียวกัน



# ตัวอย่าง



## Greedy Strategy

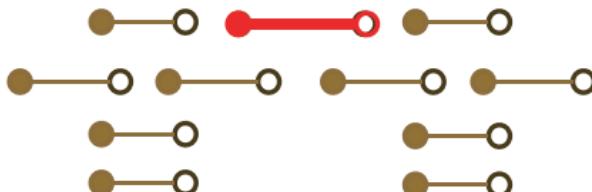
รีบใช้ : เลือก  $r_i$  น้อยสุด



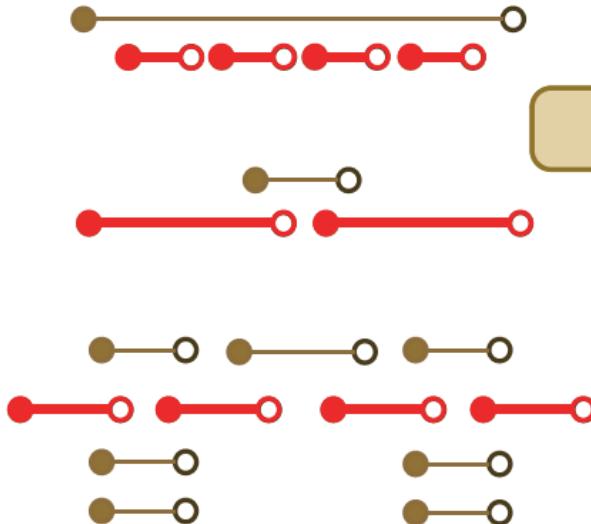
ใช้ช่วงเวลาอย่างๆ : เลือก  $f_i - s_i$  น้อยสุด



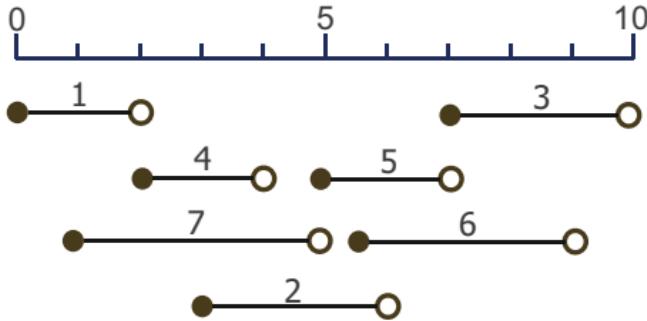
เลือกงานที่ແย่งไข้ห้องกับงานอื่นน้อยสุด



# เลือก $f_i$ น้อยสุด : เสร็จเร็วสุด



## เลือก $f_i$ น้อยสุด



เรียงลำดับตาม  $f_i$

เลือกงานที่มี  $f_i$  น้อยสุด

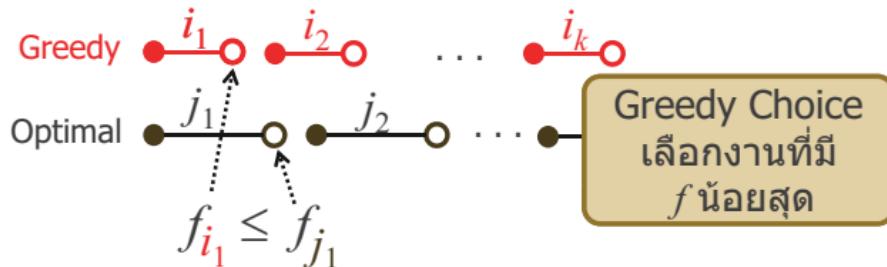
แต่ไม่สนใจงานที่มีเวลาใช้ห้องเหลือกับงานที่ได้เลือกไว้

# Greedy Activity Selection

```
GreedyActivitySelect( A[1..n] ) {  
    sort A by finish times ← O( n log n )  
    S = {1}  
    j = 1;  
    for( i = 2; i <= n; i++) { ← Θ( n )  
        if ( A[i].s >= A[j].f ) {  
            S = S ∪ { i }  
            j = i  
        }  
    }  
    return S  
}
```

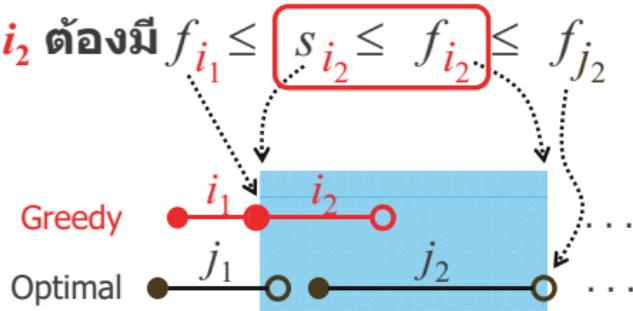
# Greedy Choice ได้ optimal soln.

- ❖ ให้ greedy เลือก  $i_1, i_2, i_3, \dots, i_k$  (เขียนเรียงตาม  $f$ )
- ❖ ให้ optimal soln คือ  $j_1, j_2, j_3, \dots, j_m$  (เขียนเรียงตาม  $f$ )
- ❖ ต้องการแสดงว่า  $k = m$
- ❖ จะแสดงให้เห็นว่า สามารถแปลง  $j_1, j_2, j_3, \dots, j_m$  เป็น  $i_1, i_2, i_3, \dots, i_k$  ได้
- ❖ ถ้า  $i_1 \neq j_1$  สามารถเปลี่ยน  $j_1, j_2, j_3, \dots, j_m$  เป็น  $i_1, j_2, j_3, \dots, j_m$  ได้

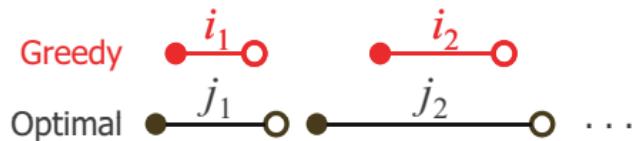


# Greedy Choice ได้ optimal soln.

- ถ้า  $i_2 \neq j_2$   $i_2$  ต้องมี  $f_{i_1} \leq f_{i_2} \leq f_{j_2}$

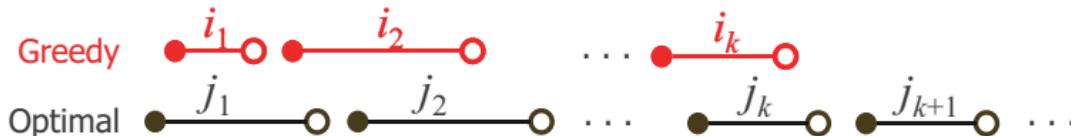


- สามารถเปลี่ยน  $i_1, j_2, j_3, \dots, j_m$  เป็น  $i_1, i_2, j_3, \dots, j_m$  ได้



# Greedy Choice ได้ optimal soln.

- ❖ สามารถเปลี่ยน  $j_1, j_2, j_3, \dots, j_k$  เป็น  $i_1, i_2, i_3, \dots, i_k$
- ❖ เป็นไปไม่ได้ที่จะมี  $j_{k+1}$  ใน optimal soln.  
 เพราะถ้ามี greedy ต้องเลือกด้วย
- ❖ ดังนั้น  $k$  ต้องเท่ากับ  $m$
- ❖ greedy choice จึงได้จำนวนงานที่ใช้ห้อง  
 เท่ากับของ optimal solution



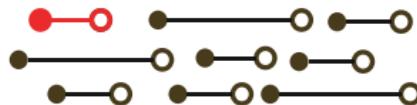
# คุณสมบัติของปัญหาที่ใช้ greedy ได้

## ❖ Optimal Substructures

- ❖ คำตอบที่ดีสุดของปัญหาใหญ่ประกอบด้วย  
    คำตอบที่ดีสุดของปัญหาย่อย  
    (ใช้ dynamic prog. ได้ ก็ต้องมีคุณสมบัติข้อนี้)

## ❖ Greedy choice

- ❖ สิ่งที่ดีสุดที่ถูกเลือก ณ ปัจจุบัน เป็นส่วนหนึ่งของ  
    คำตอบสมบูรณ์ที่ดีสุด
- ❖ dynamic prog. : แก้ทุกปัญหาย่อย แล้วค่อยนำผลมาเลือก
- ❖ greedy : เลือกแบบ greedy และ ทำให้ปัญหาเล็กลง  
    จากนั้นแค่หาคำตอบของปัญหาย่อยนั้น (ปัญหาเดียว) ก็พอ

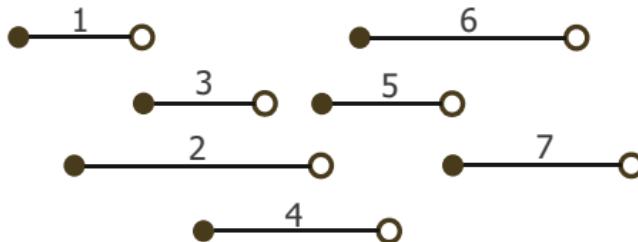


# ถ้ามีห้องจัดงานหลายห้อง

ต้องการจัดให้ได้ทุกงาน

โดยใช้ห้องจัดงานจำนวนน้อยสุด

จงออกแบบอัลกอริทึมสำหรับปัญหานี้



ลองทำดู

# ถ้ากำหนดเวลาช่วงเวลา

- ❖ งาน  $i$  ใช้ห้องเป็นเวลา  $t_i$
- ❖ ถ้ามีห้องเดียว จะจัดลำดับงานอย่างไร เพื่อให้เวลาทำงานเลิกโดยเฉลี่ยน้อยสุด

$$t_i = \text{---} 2 \text{---} 5 \text{---} 3 \text{---} 4 \text{---}$$



$$(\Sigma f) / n = (2 + 7 + 10 + 14) / 4 = 8.25$$



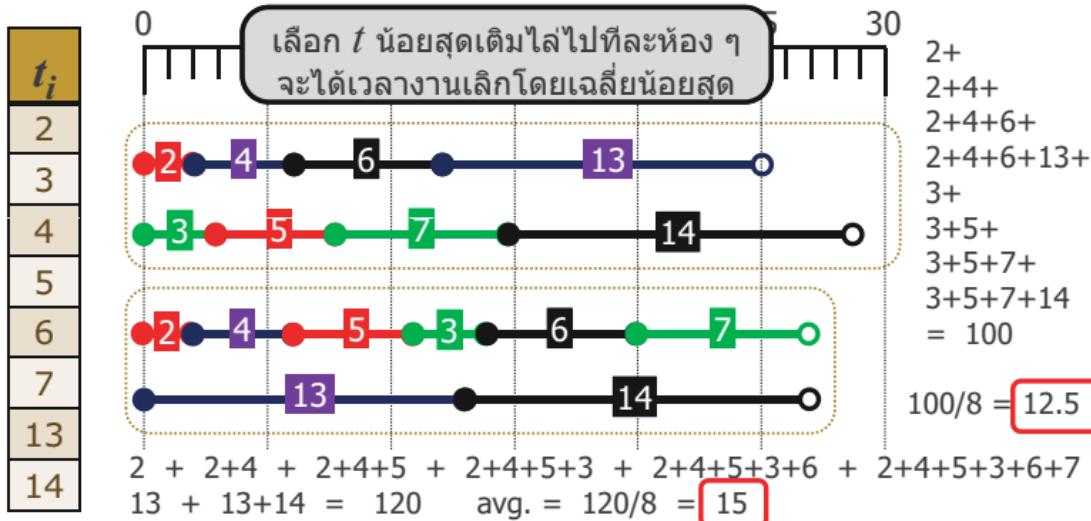
จงพิสูจน์

$$(\Sigma f) / n = (2 + 5 + 9 + 14) / 4 = 7.5$$

เลือก  $t$  น้อยสุดก่อน ได้เวลาเลิกโดยเฉลี่ยน้อยสุด

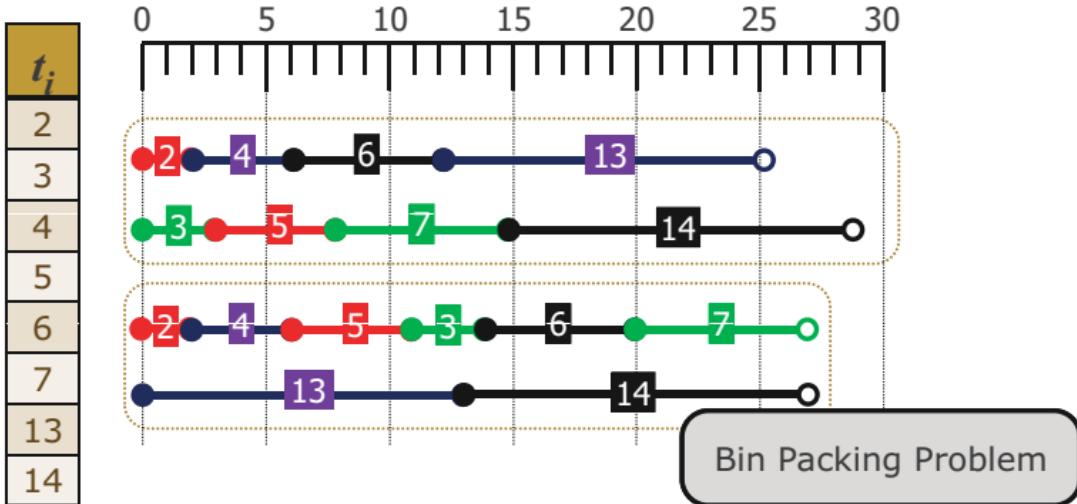
# ถ้ามีห้องห้อง

- งาน  $i$  ใช้ห้องเป็นเวลา  $t_i$
- ถ้ามี  $m$  ห้อง จะจัดลำดับงานอย่างไร ในห้องใด เพื่อให้เวลาทำงานเลิกโดยเฉลี่ยน้อยสุด



**ถ้าต้องการงานท้ายเลิกเร็วสุด**

- ❖ เวลางานเลิกโดยเฉลี่ยน้อยสุด : ง่าย
    - ❖ ใช้ greedy choice : เลือก  $t$  น้อยสุดเติม "ໄล" ไปทีละห้อง ๆ
  - ❖ เวลาเลิกของงานท้ายสุด น้อยสุด : ยาก



# ปัญหา Fractional Knapsack

- ❖ ของ  $n$  ชิ้นมีหมายเลข :  $1, 2, 3, \dots, n$
- ❖ แต่ละชิ้นหนัก :  $w_1, w_2, w_3, \dots, w_n$
- ❖ แต่ละชิ้นมีมูลค่า :  $v_1, v_2, v_3, \dots, v_n$
- ❖ ถุงเป้หนึ่งใบจุของได้หนักไม่เกิน  $W$
- ❖ ปัญหา : จงเลือกของใส่ถุง เพื่อให้
  - ❖ ถุงไม่ขาด
  - ❖ ได้มูลค่ารวมมากสุด
  - ❖ หมายเหตุ : อนุญาตให้เฉือน (แบ่ง) ของบางส่วนได้ ได้มูลค่าเพรตตามน้ำหนัก



# Optimization Problem

- ❖ ของ  $n$  ชิ้นมีหมายเลข :  $1, 2, 3, \dots, n$
- ❖ แต่ละชิ้นหนัก :  $w_1, w_2, w_3, \dots, w_n$
- ❖ แต่ละชิ้นมีมูลค่า :  $v_1, v_2, v_3, \dots, v_n$
- ❖ ถุงเป็นหนึ่งในจุดของได้นักไม่เกิน  $W$
- ❖ หา  $\langle x_1, x_2, x_3, \dots, x_n \rangle$ ,  $x_k = 0$  ถึง 1

$$\text{maximize} \sum_{k=1}^n x_k v_k$$

maximization  
problems

$$\text{subject to } \sum_{k=1}^n x_k w_k \leq W$$

$$0 \leq x_k \leq 1$$

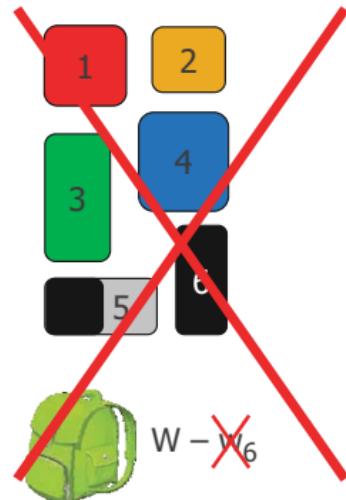
# Knapsack มี Optimal substruct.



W



W - w<sub>6</sub>



W - ~~w<sub>6</sub>~~

$$v_1 + v_3 + v_4 + v_6$$

$$v_1 + v_3 + v_4$$

$$v_1 + v_2 + v_4 + v_5/2 + v_6$$

เป็นไปไม่ได้ที่จะมากกว่า

คำตอบดีสุดของปัญหาใหญ่  
มีคำตอบดีสุดของปัญหาย่อย

# Greedy Choice

เลือกของแพง

$$W = 100$$

$$\begin{array}{l} v_1=66 \\ w_1=30 \end{array}$$

$$\begin{array}{l} v_2=20 \\ w_2=10 \end{array}$$

$$\begin{array}{l} v_3=30 \\ w_3=20 \end{array}$$

$$\begin{array}{l} v_4=60 \\ w_4=50 \end{array}$$

$$\begin{array}{l} v_5=40 \\ w_5=40 \end{array}$$

$$\Sigma v = 146$$

$$\Sigma w = 30+50+20$$

เลือกของเบา

$$\begin{array}{l} v_1=66 \\ w_1=30 \end{array}$$

$$\begin{array}{l} v_2=20 \\ w_2=10 \end{array}$$

$$\begin{array}{l} v_3=30 \\ w_3=20 \end{array}$$

$$\begin{array}{l} v_4=60 \\ w_4=50 \end{array}$$

$$\begin{array}{l} v_5=40 \\ w_5=40 \end{array}$$

$$\Sigma v = 156$$

$$\Sigma w = 10+20+30+40$$

เลือกของคุ้ม (มูลค่าต่อน้ำหนัก)

$$\begin{array}{l} v_1=66 \\ w_1=30 \end{array}$$

$$\begin{array}{l} v_2=20 \\ w_2=10 \end{array}$$

$$\begin{array}{l} v_3=30 \\ w_3=20 \end{array}$$

$$\begin{array}{l} v_4=60 \\ w_4=50 \end{array}$$

$$\begin{array}{l} v_5=40 \\ w_5=40 \end{array}$$

$$\Sigma v = 164$$

$$\Sigma w = 30+10+20+40$$

$$v/w : 2.2$$

$$2.0$$

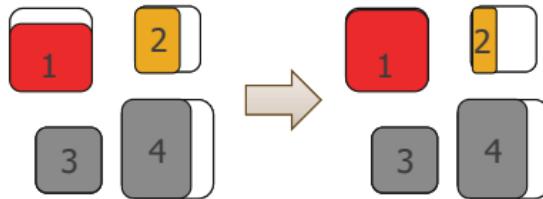
$$1.5$$

$$1.2$$

$$1.0$$

# Greedy Choice ได้ Optimal Soln.

$$v_1/w_1 \geq v_2/w_2 \geq \dots \geq v_n/w_n$$



ถ้าเลือกชิ้น 1 ไม่เต็มชิ้น ( $x_1 < 1$ ) และมีการเลือกชิ้น  $k$  อื่น ( $x_k > 0$ )

- ลดชิ้น  $k$  ลง  $\Delta$   $\Delta = \min(x_k, 1 - x_1)$
- เพิ่มชิ้น 1 ขึ้น  $\Delta$
- จะได้มูลค่ารวมมากขึ้น  $\left(\frac{v_1}{w_1} - \frac{v_k}{w_k}\right)\Delta \geq 0$
- การเลือกชิ้นที่มี  $v_1/w_1$  ให้มากสุด จึงถูกต้อง

# Greedy Knapsack

```

GreedyKnapsack( A[1..n], w ) {
    sort A by v/w มากไปน้อย ← O( n log n )
    sumW = 0
    for( i = 1; i <= n AND sumW < w; i++ ) {
        A[i].x = min(A[i].w, W - sumW)/A[i].w
        sumW += A[i].x * A[i].w ← O( n )
    }
    return A
}

```

$v_1 = 66$

$w_1 = 30$

$v_2 = 20$

$w_2 = 10$

$v_3 = 30$

$w_3 = 20$

$v_4 = 60$

$w_4 = 50$

$v_5 = 40$

$w_5 = 40$

$\sum v = 164$

$\sum w = 30+10+20+40$

$v/w : 2.2$

$2.0$

$1.5$

$1.2$

$1.0$

## เลือกคุ้มสุด : ได้มูลรวมสูงสุด

- ❖ ให้  $v_1/w_1 \geq v_2/w_2 \geq \dots \geq v_n/w_n$
- ❖ ให้  $x$  เป็น คำตอบที่ได้จาก greedy choice
- ❖ ให้  $y$  เป็น คำตอบใด ๆ ที่เป็นไปได้ (ถุงไม่ขาด)
- ❖ ต้องแสดงให้เห็นจริงว่า

$$\sum_{i=1}^n x_i v_i \geq \sum_{i=1}^n y_i v_i$$

$$\sum_{i=1}^n (x_i - y_i) v_i \geq 0$$

# เลือกคุ้มสุด : ได้รวมสูงสุด

$\begin{matrix} & 1 & 2 & & k & & n \\ x & \boxed{1} & \boxed{1} & \dots & \boxed{1} & \boxed{x_k} & \boxed{0} & \dots & \boxed{0} \end{matrix}$ 
 $x_k < 1$

$$\sum_{i=1}^n (x_i - y_i) v_i = \left\{ \begin{array}{l}
 \sum_{i=1}^{k-1} (x_i - y_i) w_i \frac{v_i}{w_i} \\
 + (x_k - y_k) w_k \frac{v_k}{w_k} \\
 + \sum_{i=k+1}^n (x_i - y_i) w_i \frac{v_i}{w_i}
 \end{array} \right\} \geq \sum_{i=1}^{k-1} (x_i - y_i) w_i \frac{v_k}{w_k} = (x_k - y_k) w_k \frac{v_k}{w_k} \geq \sum_{i=k+1}^n (x_i - y_i) w_i \frac{v_k}{w_k}$$

# ເລືອກຄຸ້ມສຸດ : ໄດ້ມູນລຽມສູງສຸດ

$$x \quad \begin{array}{ccccccccc} 1 & 2 & & k & & n \\ \boxed{1} & \boxed{1} & \dots & \boxed{1} & \boxed{x_k} & \boxed{0} & \dots & \boxed{0} \end{array} \quad x_k < 1$$

$$\sum_{i=1}^n (x_i - y_i) v_i = \left\{ \begin{array}{l} \sum_{i=1}^{k-1} (x_i - y_i) w_i \frac{v_i}{w_i} \\ + (x_k - y_k) w_k \frac{v_k}{w_k} \\ + \sum_{i=k+1}^n (x_i - y_i) w_i \frac{v_i}{w_i} \end{array} \right\} \geq \sum_{i=1}^n (x_i - y_i) w_i \frac{v_k}{w_k}$$

$$\sum_{i=1}^n x_i w_i = W \quad \sum_{i=1}^n y_i w_i \leq W$$

# Greedy ใช้กับ 0/1 Knapsack ไม่ได้

$$W = 25$$

Fractional  
Knapsack

$$v/w : 6.0$$

$$v_1=30$$

$$w_1=5$$

$$v_2=50$$

$$w_2=10$$

$$v_3=60$$

$$w_3=15$$

$$\Sigma v = 120$$

$$\Sigma w = 5+10+10$$

0/1  
Knapsack

$$v_1=30$$

$$w_1=5$$

$$v_2=50$$

$$w_2=10$$

$$v_3=60$$

$$w_3=15$$

$$\Sigma v = 80$$

$$\Sigma w = 15$$

$$v_1=30$$

$$w_1=5$$

$$v_2=50$$

$$w_2=10$$

$$v_3=60$$

$$w_3=15$$

$$\Sigma v = 90$$

$$\Sigma w = 20$$

$$v_1=30$$

$$w_1=5$$

$$v_2=50$$

$$w_2=10$$

$$v_3=60$$

$$w_3=15$$

$$\Sigma v = 110$$

$$\Sigma w = 25$$

# 0/1 Knapsack $\leftrightarrow$ Scheduling

## ❖ Scheduling

- ❖ จัด  $n$  งานที่ใช้เวลา  $t_1, t_2, \dots, t_n$
- ❖ ให้กับห้อง  $m$  ห้อง  $m = 2$
- ❖ ต้องการเวลาเลิกของงานท้ายสุด น้อยสุด



$t_i$
2
3
4
5
6
7
13
14

## ❖ เหมือนกับปัญหา 0/1 knapsack

- ❖ ที่มีของ  $n$  ชิ้น
- ❖ มีน้ำหนัก  $t_1, t_2, \dots, t_n$   $W = 54 / 2$
- ❖ มีมูลค่าเท่ากันหมด  $= 27$
- ❖ ถุงเบร์น้ำหนักไม่เกิน  $(t_1 + t_2 + \dots + t_n) / 2$   $\{2, 3, 4, 5, 6, 7\}$

# รหัสข้อมูล

## ❖ จำนวนเต็ม : two's complement

- ❖ 23 → 00000000000010111
- ❖ -23 → 1111111111101001

## ❖ ข้อความ : รหัส Unicode

- ❖ A → 00000000001000001
- ❖ ก → 00001110000000001
- ❖ 吳 → 0101010000110011

## ❖ รหัสความยาวคงที่

## ❖ รหัสความยาวแปรไป

- ❖ ข้อมูลความถี่สูง ใช้รหัสสั้น
- ❖ ข้อมูลความถี่ต่ำ ใช้รหัสยาว

# รหัส Huffman

จีกจีกงักก้มันเป็นจีกจีกงัก  
จีกจีกงักก้มันเป็นจีกจีกงัก  
มันเป็นกะอึกกะอักมันเป็นจีกจีกจีกจีก

ก	ง	ـ	ـ	ນ	ຈ	ປ	ນ	ໄ	ـ	ອ	ະ	ـ
24	16	15	11	8	4	4	4	4	4	2	2	2
0000	0001	0010	1010	0011	0100	0101	0110	0111	1000	1001	1011	1100
11	010	000	001	0110	01110	01111	1000	1001	10100	10101	10110	10111

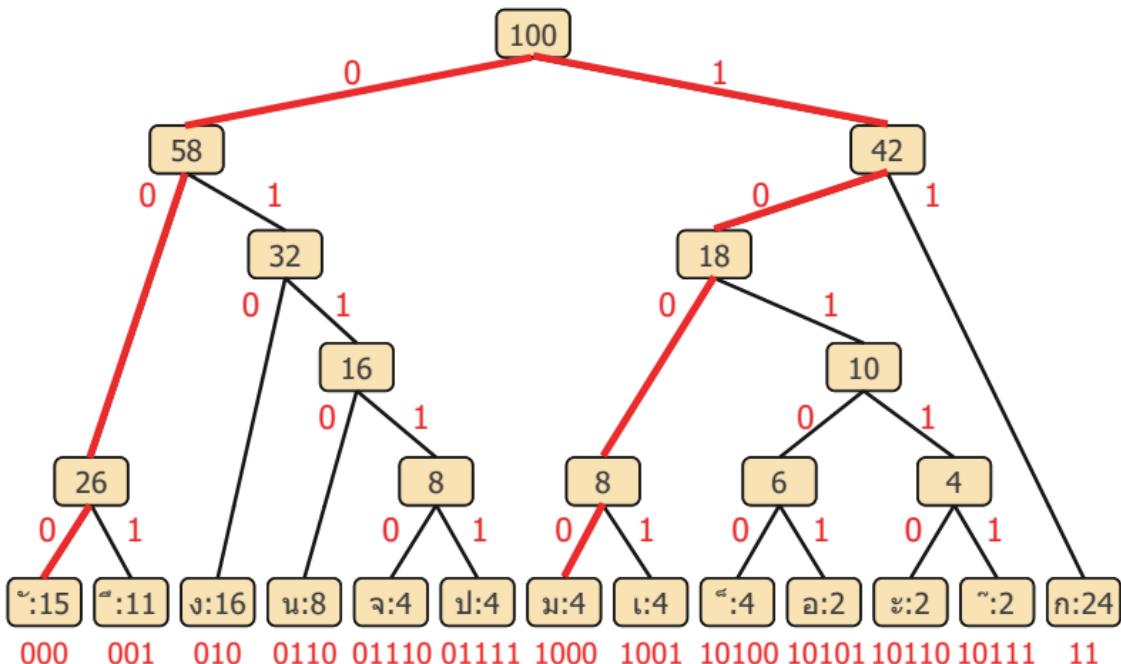
$$4 \times (24 + 16 + 15 + 11 + 8 + 4 + 4 + 4 + 4 + 4 + 2 + 2 + 2) = 400 \text{ มิต}$$

variable-length code  
prefix-free code

$$\left. \begin{array}{l} 2 \times 24 + \\ 3 \times (16 + 15 + 11) + \\ 4 \times (8 + 4 + 4) + \\ 5 \times (4 + 4 + 4 + 2 + 2 + 2) \end{array} \right\} = 328 \text{ มิต}$$

ไม่มีรหัสใดเป็นส่วนหนึ่งของรหัสอื่น

# การหารหัส Huffman



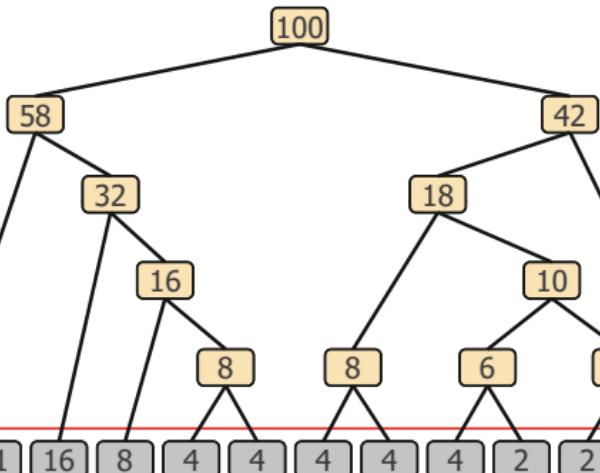
# Huffman Tree

```
HuffmanTree( C[1..n] )
    minHeap = new min heap
    for (i = 1; i <= n; i++) {
        minHeap.add( new Node(C[i].code, C[i].freq) )
    }
    for ( i = 1; i < n; i++ ) {
        node = new Node( )
        node.left = minHeap.removeMin()
        node.right = minHeap.removeMin()
        node.freq = node.left.freq + node.right.freq
        minHeap.add( node )
    }
    return minHeap.removeMin()
}
```

$O(n \log n)$

$O(n \log n)$

# Huffman Tree ตีสุด



Huffman tree  
มี  $B(T)$  น้อยสุด

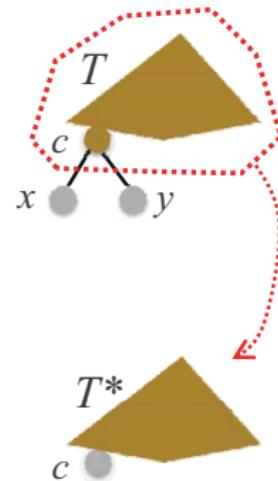
$$B(T) = \sum_{c \in C} f(c)d_T(c)$$

$c$  รหัส  
 $f(c)$  ความถี่ของ  $c$   
 $d_T(c)$  ความลึกของปม  $c$  ในต้น  $T$   
 (จำนวนบิตของรหัส  $c$ )

# Huffman Tree มี $B(T)$ น้อยสุด

## ❖ Greedy-choice

- ❖ เลือกปัมที่มีความถี่น้อยสุดมารวมก่อน จะได้ตันที่มี  $B(T)$  น้อยสุด
- ❖ มีตันที่  $B(T)$  น้อยสุด ที่มีปัมความถี่ น้อยสุดอยู่ระดับล่างสุด



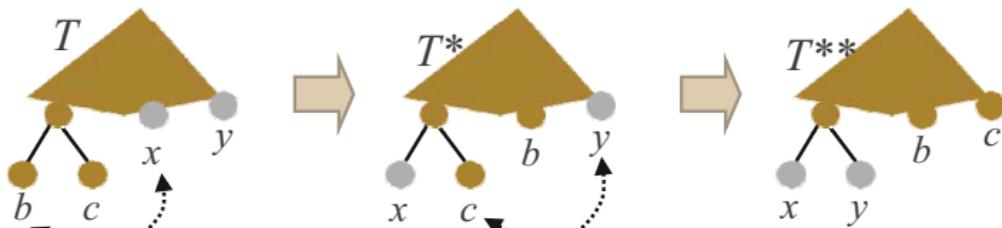
## ❖ Optimal-substructure

- ❖ ตันที่  $B(T)$  น้อยสุด ประกอบด้วย ตันไม้มายอยที่มี  $B(T^*)$  น้อยสุดด้วย

$$B(T) = \sum_{c \in C} f(c)d_T(c)$$

# Greedy Choice ได้ Optimal Soln

ให้  $x$  และ  $y$  มี  $f(x)$  และ  $f(y)$  น้อยสุด



$$\sum_{c \in C} f(c)d_T(c) - \sum_{c \in C} f(c)d_{T^*}(c)$$

ต้นที่มี  $x$  และ  $y$  อยู่คู่กัน  
ล่างสุดเป็น optimal soln

$$= f(x)d_T(x) + f(b)d_T(b) - f(x)d_{T^*}(x) - f(b)d_{T^*}(b)$$

$$= f(x)d_T(x) + f(b)d_T(b) - f(x)d_T(b) - f(b)d_T(x)$$

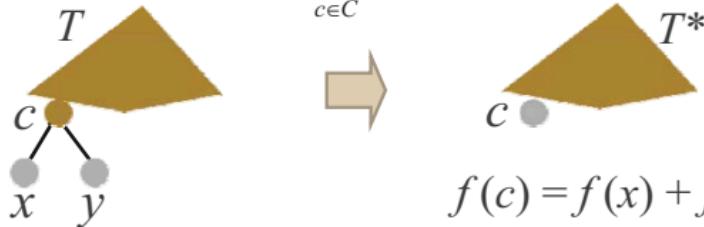
$$= (f(x) - f(b))d_T(x) + (f(b) - f(x))d_T(b)$$

$$= (f(b) - f(x))(d_T(b) - d_T(x)) \geq 0$$

$B(T) \geq B(T^*)$

# มี Optimal Substructure

$$B(T) = \sum_{c \in C} f(c)d_T(c)$$



$$B(T) = B(T^*) + (f(x)d_T(x) + f(y)d_T(y)) - f(c)d_{T^*}(c)$$

$$B(T) = B(T^*) + f(x) + f(y)$$

$$f(x)d_T(x) + f(y)d_T(y) = (f(x) + f(y))(d_{T^*}(c) + 1)$$

$$= f(c)d_{T^*}(c) + f(x) + f(y)$$

$B(T)$  จะน้อยสุดได้,  
 $B(T^*)$  ต้องน้อยสุด

# Optimal 2-Way Merge

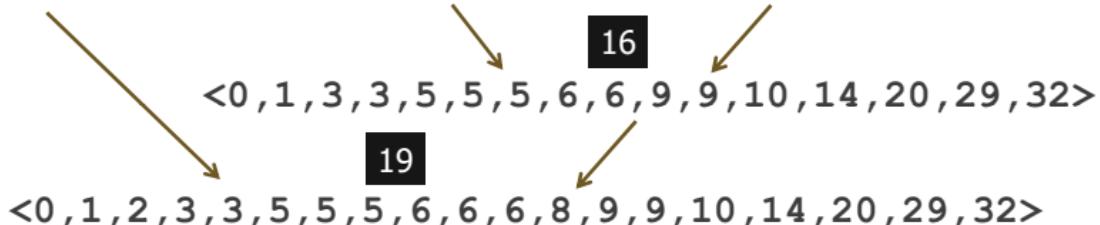
$\langle 2, 6, 8 \rangle \quad \langle 1, 3, 5, 6, 9, 20 \rangle \quad \langle 0, 3, 5, 5, 6, 9, 10, 14, 29, 32 \rangle$



$\langle 0, 1, 2, 3, 3, 5, 5, 5, 6, 6, 6, 8, 9, 9, 10, 14, 20, 29, 32 \rangle$

หาลำดับการทำ 2-way merge ให้ได้จำนวนการย้ายข้อมูลน้อยสุด

$\langle 2, 6, 8 \rangle \quad \langle 1, 3, 5, 6, 9, 20 \rangle \quad \langle 0, 3, 5, 5, 6, 9, 10, 14, 29, 32 \rangle$



ลองทำเอง คล้าย ๆ การหา Huffman tree

# อัลกอริทึมแบบละไมบอีน ๆ

## ❖ การหาวิถีสั้นสุด

- ❖ อัลกอริทึมของดิกสตรา (Dijkstra)

## ❖ การหาต้นไม้แบบหอดข้ามตัวสุด

- ❖ อัลกอริทึมของพริม (Prim)
- ❖ อัลกอริทึมของครูสกัล (Kruskal)



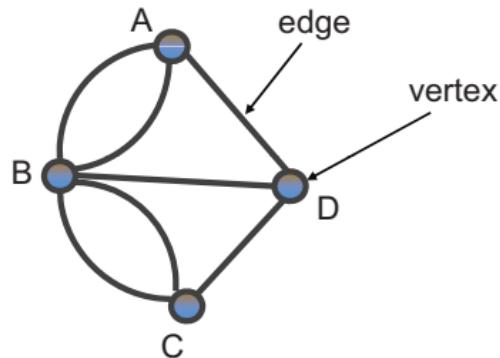
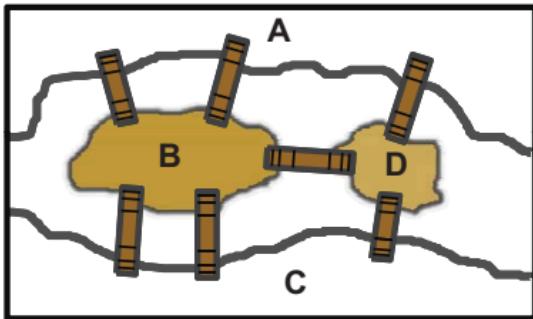
# กราฟเบื้องต้น

สมชาย ประสิทธิ์จูตราภูมิ

# หัวข้อ

นิยาม  
ตัวอย่างการใช้กราฟ  
กราฟประเภทต่าง ๆ  
วิถีและวง  
การแทนกราฟ  
การແວ່ຜ່ານ  
ສພາກເຮືອມໂຍງຂອງกรາฟ

# กราฟ (Graph)



Königsberg Bridge

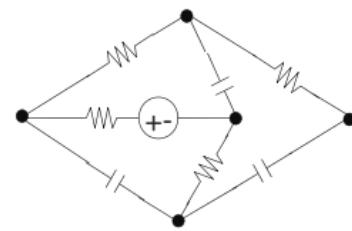
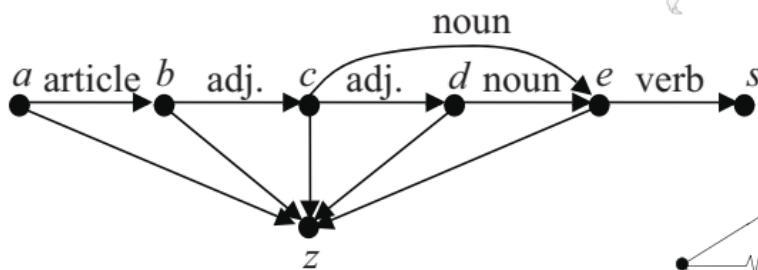
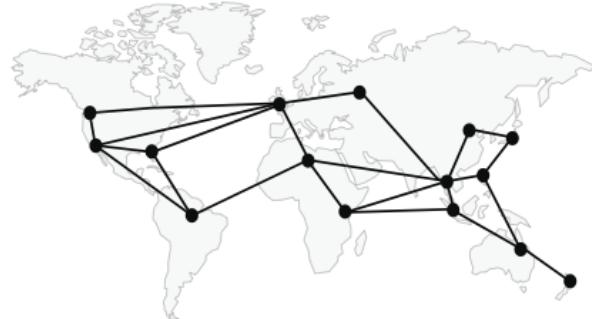
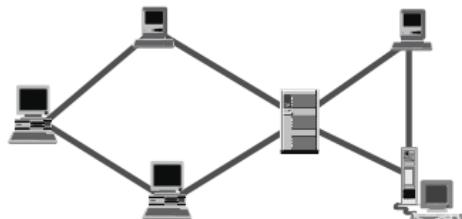
Problem

1736:

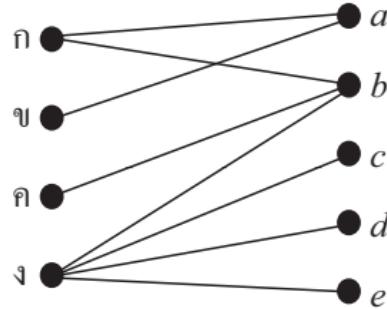
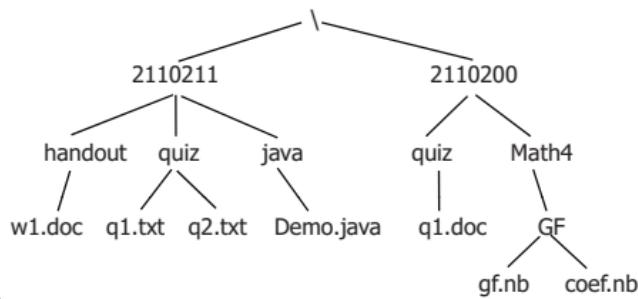
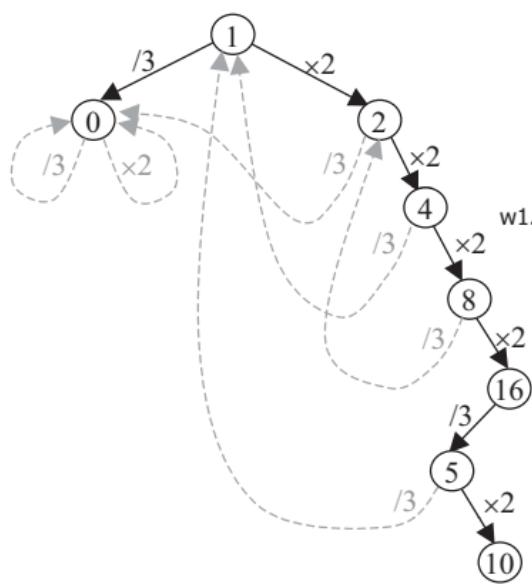
Leonhard Euler



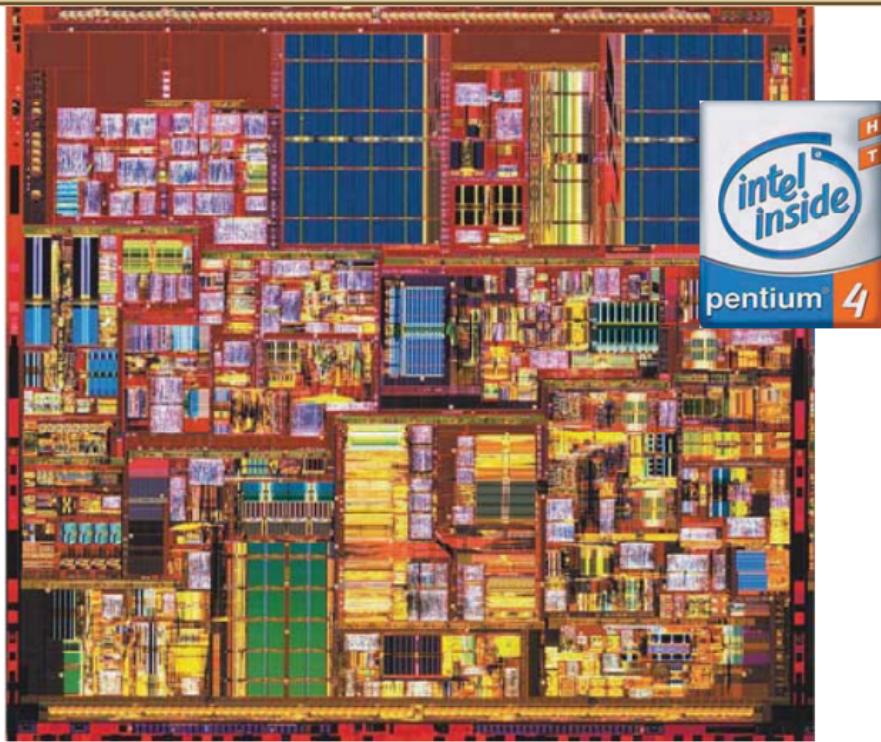
# ใช้กราฟแทน ...



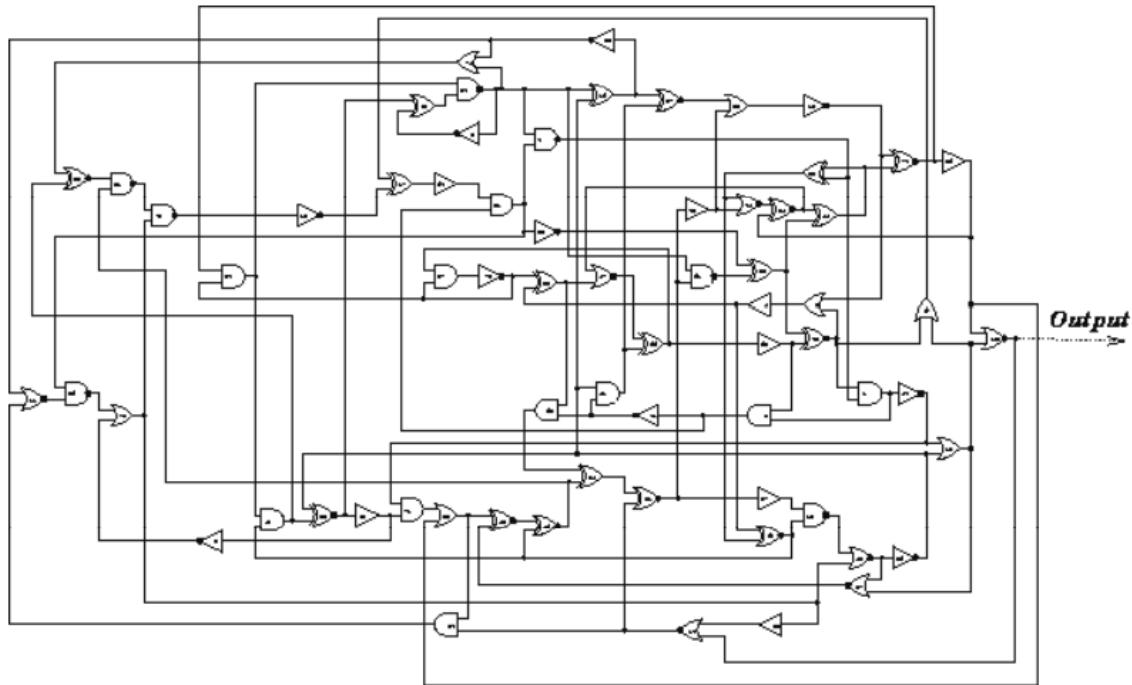
# ใช้กราฟแทน ...



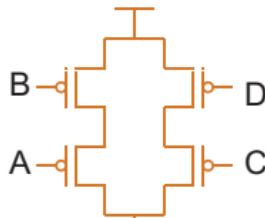
# กราฟกับการออกแบบวงจรรวม



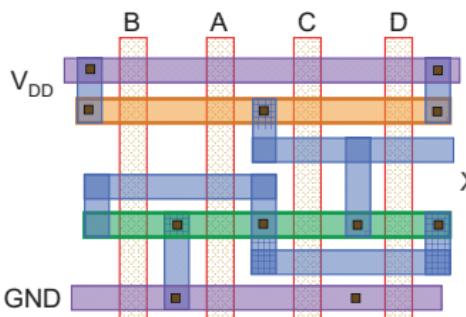
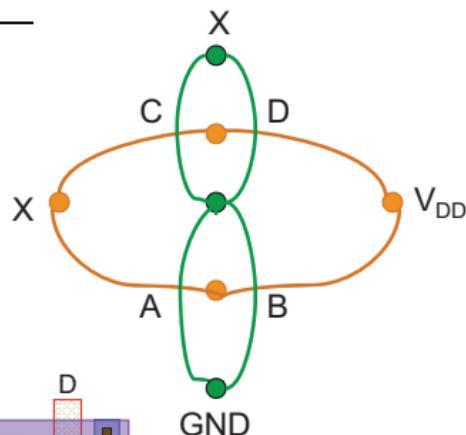
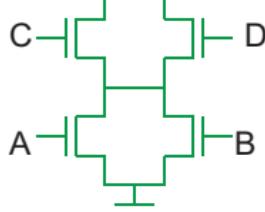
# กราฟกับวงจรดิจิตอล



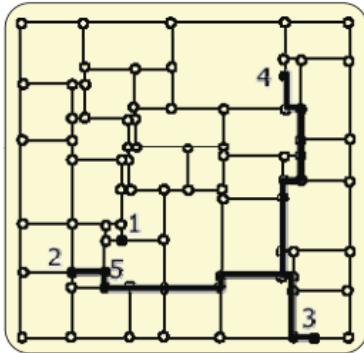
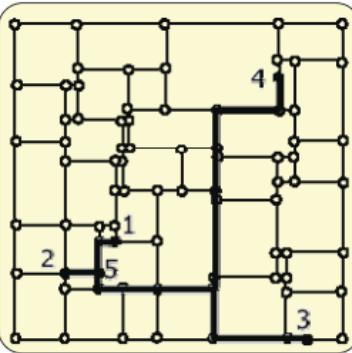
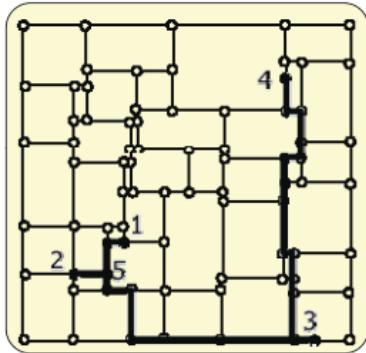
# กราฟกับการออกแบบวงจร



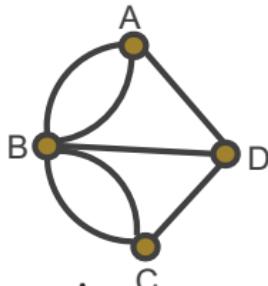
$$X = \overline{(A+B) \cdot (C+D)}$$



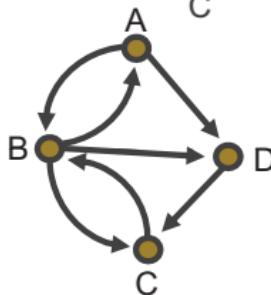
# กราฟกับการเชื่อมสายสัญญาณ



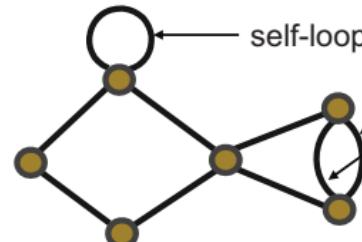
# กราฟประเภทต่าง ๆ



undirected graph

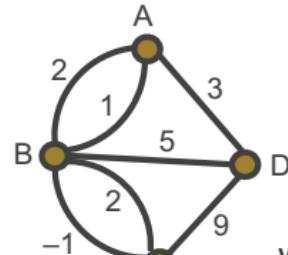


directed graph

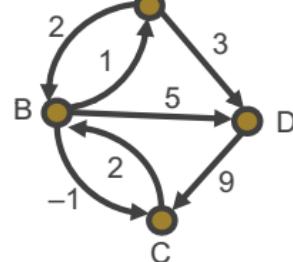


parallel  
edges

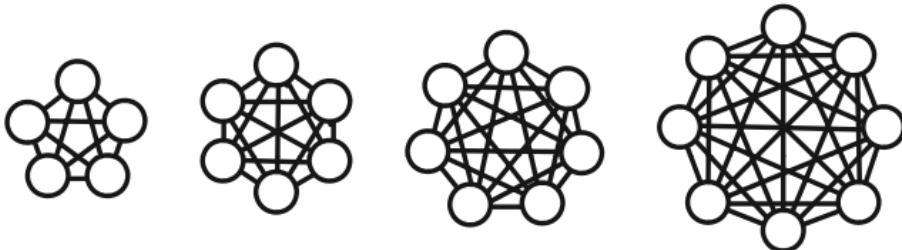
simple graph ไม่มี self-loop และ parallel edges



weighted graph



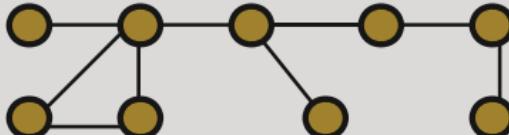
# Complete Graphs



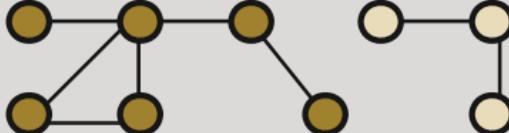
complete graph ที่มี  $v$  vertices มี  $v(v - 1)/2$  edges

simple graph :  $e = O(v^2)$

# Connected Graphs



กราฟต่อถึงกันมี  
1 component

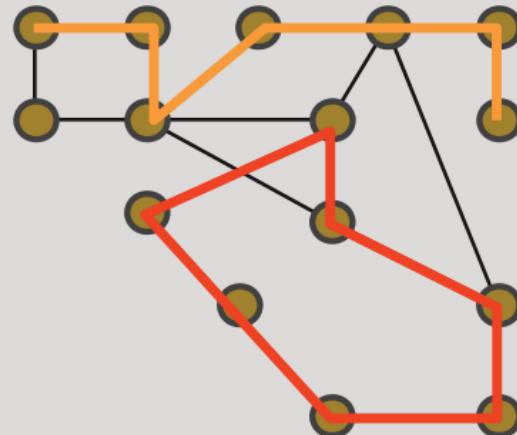


2 components

กราฟต่อถึงกันที่มี  $v$  ปม  
มีเส้นเชื่อมอย่างน้อย  $v - 1$  เส้น

connected graph  
 $e = \Omega(v)$

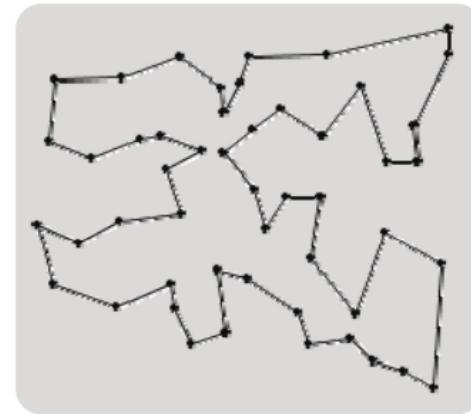
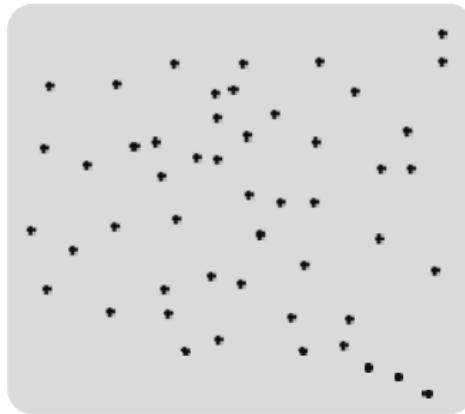
# Path (วิถี) & Cycle (วง)



Simple path หรือ simple cycle คือวิถี  
หรือวงที่ผ่านปมละอย่างมากหนึ่งครั้ง

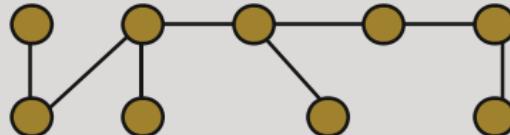
# Traveling Salesperson Problem

ให้ปัมແທນເມືອງ ມີຄົນຈາກທຸກເມືອງໄປທຸກເມືອງ  
ພັນກົງນາຂາຍຕ້ອງເດີນທາງແວະຜ່ານທຸກເມືອງອຍ່າງໄຣ  
ໃຫມໍການເດີນທາງທີ່ສັນສົດໂດຍໄມ່ຜ່ານເມືອງຊ້າ



# Trees : ต้นไม้คือ...

- ❖ กราฟต่อถึงกันที่ไม่มีวง
- ❖ กราฟที่มี  $v$  ปม มี  $v-1$  เส้น และไม่มีวง
- ❖ กราฟต่อถึงกันที่มี  $v$  ปม มี  $v-1$  เส้น
- ❖ กราฟที่มี simple path เพียงวิถีระหว่างปมทุกคู่



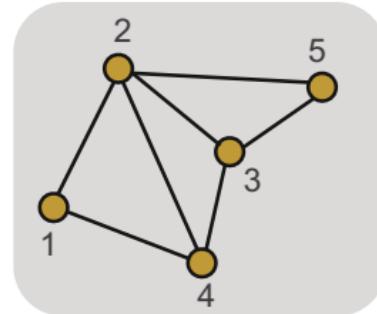
# การแทนกราฟ

## ❖ adjacency matrix

	1	2	3	4	5
1	0	1	0	1	0
2	1	0	1	1	1
3	0	1	0	1	1
4	1	1	1	0	0
5	0	1	1	0	0

## ❖ adjacency list

1	→ < 2, 4 >
2	→ < 1, 3, 4, 5 >
3	→ < 2, 4, 5 >
4	→ < 1, 2, 3 >
5	→ < 2, 3 >



# คำถามที่พบบ่อยของกราฟ

## ❖ adjacency matrix

	1	2	3	4	5
1	0	1	0	1	0
2	1	0	1	1	1
3	0	1	0	1	1
4	1	1	1	0	0
5	0	1	1	0	0

มีเส้นเชื่อมต่อระหว่าง  
ปม a กับ b หรือไม่ ?

$\Theta(1)$

$\Theta(v)$

$O(v)$

$O(v)$

## ❖ adjacency list



ปม a มีเส้นเชื่อมต่อ  
กับปมใดบ้าง ?

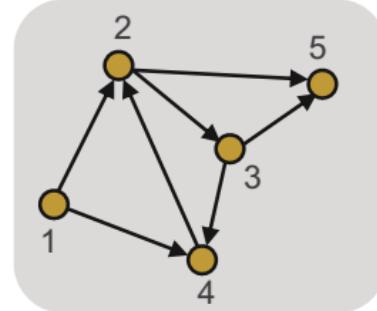
# กราฟระบุทิศทาง (Directed Graph)

## ❖ adjacency matrix

	1	2	3	4	5
1	0	1	0	1	0
2	0	0	1	0	1
3	0	0	0	1	1
4	0	1	0	0	0
5	0	0	0	0	0

## ❖ adjacency list

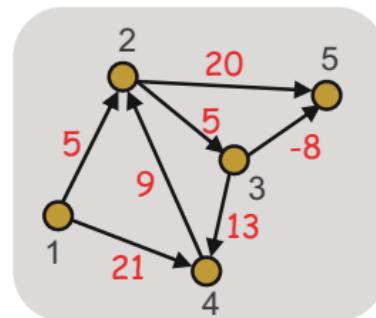
1	→ < 2, 4 >
2	→ < 3, 5 >
3	→ < 4, 5 >
4	→ < 2 >
5	→ < >



# กราฟถ่วงน้ำหนัก (Weighted Graph)

## ❖ adjacency matrix

	1	2	3	4	5
1	0	5	0	21	0
2	0	0	5	0	20
3	0	0	0	13	-8
4	0	9	0	0	0
5	0	0	0	0	0



## ❖ adjacency list

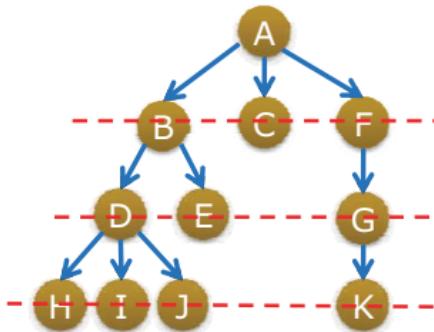
1	→ < (2, 5) , (4, 21) >
2	→ < (3, 5) , (5, 20) >
3	→ < (4, 13) , (5, -8) >
4	→ < (2, 9) >
5	→ < >

# Basic Graph Algorithms

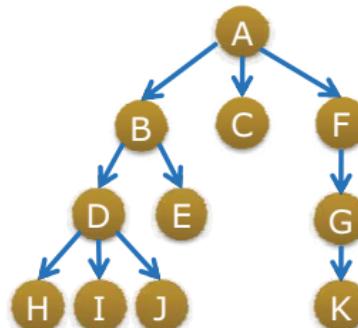
- ❖ Breadth-First Search
- ❖ Depth-First Search
- ❖ Topological Sort
- ❖ Strongly Connected Components

# การค้นต้นไม้

Breadth-first search

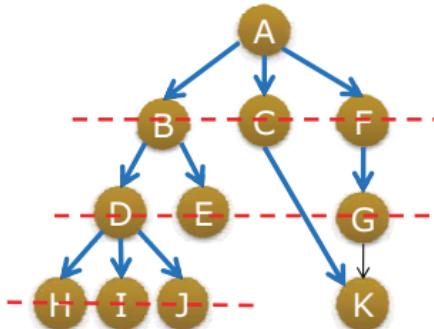


Depth-first search



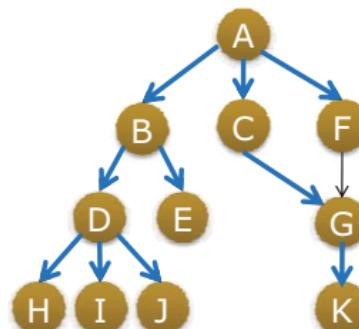
# การค้นกราฟ

Breadth-first search



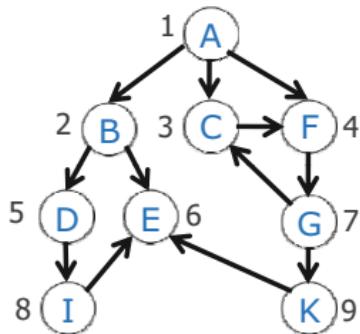
ใช้ queue จำปน  
ระหว่างการค้น

Depth-first search



ใช้ stack จำปน  
ระหว่างการค้น

# Breadth-First Search



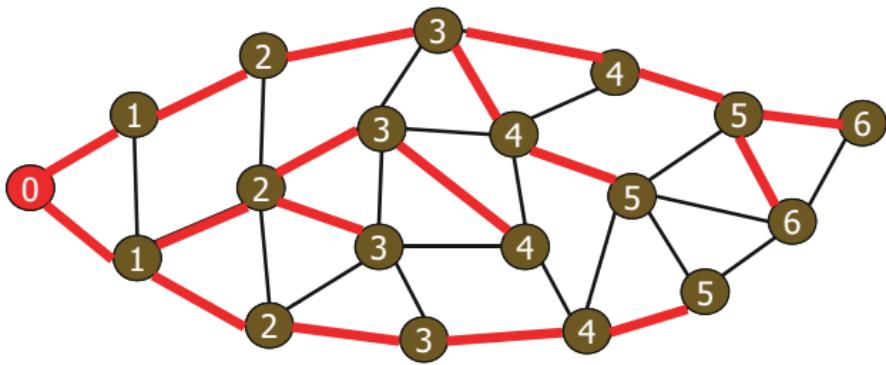
$\Theta(|V| + |E|)$

ถ้าใช้ adjacency list

```

bfs(G=(V,E), s) {
    for each v ∈ V
        color[v] ← WHITE;
    Q ← an empty queue;
    Q.enqueue(s);
    color[s] ← GRAY
    while ( Q ≠ ∅ ) {
        u ← Q.dequeue();
        color[u] ← BLACK;
        for each v ∈ adj(u)
            if (color[v]==WHITE) {
                Q.enqueue(v);
                color[v] ← GRAY
            }
    }
}
  
```

## BFS Tree : Path Finding

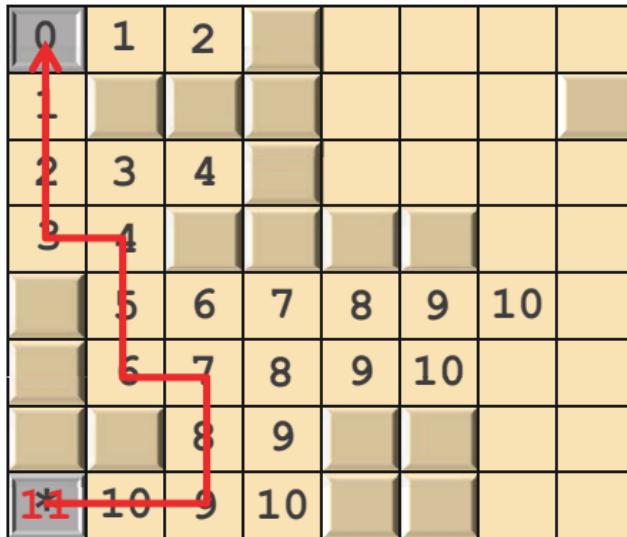


$d[u]$  คือจำนวนเส้นเชื่อมจากจุดเริ่มต้นถึง  $u$

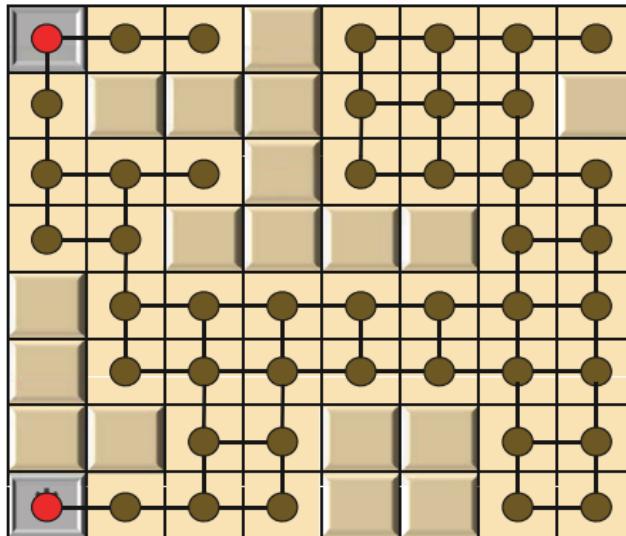
# BFS : Path Finding

```
bfs(G=(V,E), s, t) {
    for each v ∈ V
        d[v] ← ∞; p[v] ← null;
    Q ← an empty queue;
    Q.enqueue(s);
    d[s] ← 0;
    while ( Q ≠ Ø ) {
        u ← Q.dequeue();
        for each v ∈ adj(u)
            if ( d[v] == ∞ ){
                d[v] ← d[u] + 1; p[v] ← u;
                Q.enqueue(v);
            }
    }
}
```

## BFS : Shortest Path

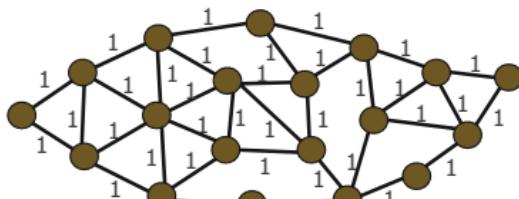


# BFS : Shortest Path

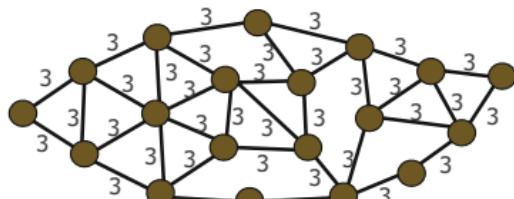


# ใช้ BFS หา Shortest Path

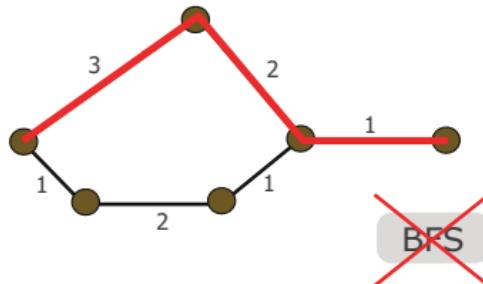
หาวิถีสั้นสุดได้ ถ้าเส้นเชื่อมทุกเส้นยาวเท่ากัน



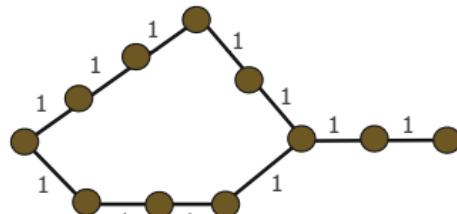
BFS



BFS



~~BFS~~



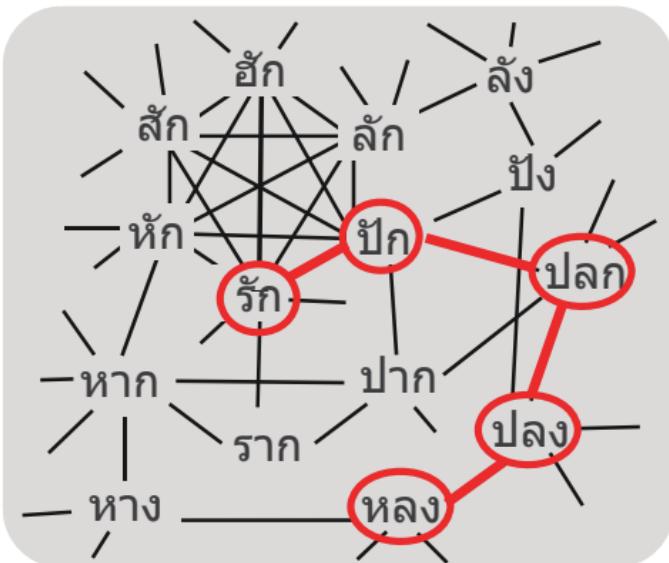
ไม่ควรทำแบบนี้  
มีวิธีอื่นที่ดีกว่า

BFS

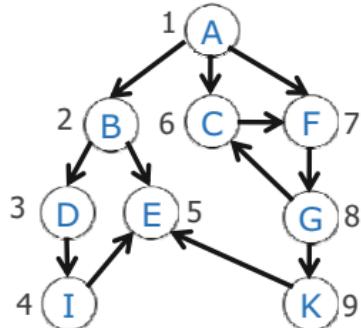
# Word Ladder

รัก → ปีก → ปลอก → ปลง → หลง

#### ต้องการการเปลี่ยนที่สั้นสุด : ใช้ BFS



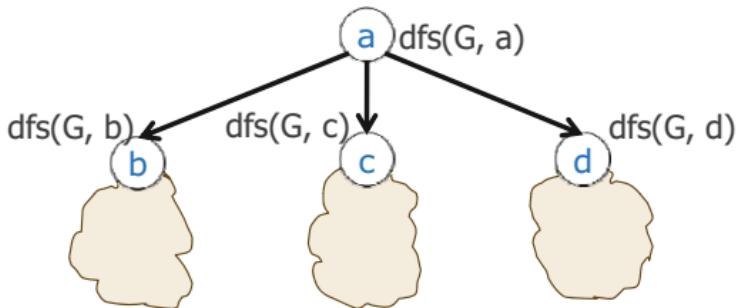
# Depth-First Search



```
dfs(G=(V,E), s) {  
    color[] ← WHITE  
    S ← an empty stack  
    color[s]←GRAY  
    S.push(s);  
    while ( S ≠ Ø ) {  
        u ← S.top()  
        choose v ∈ adj(u) AND  
            color[v] is WHITE  
        if ( v does not exist ) {  
            u ← S.pop()  
            color[u]←BLACK  
        } else {  
            color[v]←GRAY  
            S.push(v)  
        }  
    }  
}
```

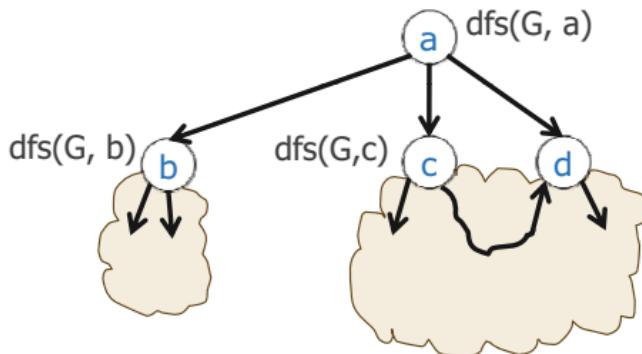
# Depth-First Search

```
dfs(G=(V,E), u) {  
    color[u] ← GRAY  
    for each v ∈ adj(u)  
        if (color[v] == WHITE)  
            dfs(G, v)  
    color[u] ← BLACK  
}
```



# Depth-First Search

```
dfs(G=(V,E), u) {  
    color[u] ← GRAY  
    for each v ∈ adj(u)  
        if (color[v] == WHITE)  
            dfs(G, v)  
    color[u] ← BLACK  
}
```



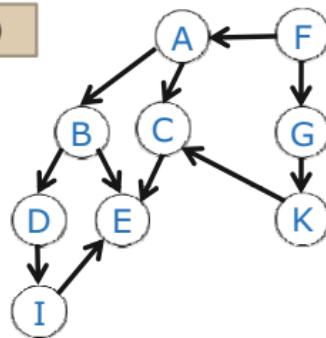
# DFS( $G, u$ ) : ไม่จำเป็นต้องผ่านทุกปม

```

dfs (G=(V,E) , u) {
    color[u] ← GRAY
    for each v ∈ adj(u)
        if (color[v] == WHITE)
            dfs(G, v)
    color[u] ← BLACK
}

```

dfs( G, A )

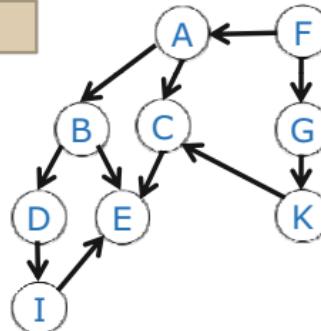


# DFS( G ) : ผ่านทุกปม

```
dfs( G=(V,E) ) {  
    for each v ∈ V  
        color[v] ← WHITE  
    for each v ∈ V  
        if (color[v] == WHITE)  
            dfs(G, v)  
}
```

```
dfs(G=(V,E), u) {  
    color[u] ← GRAY  
    for each v ∈ adj(u)  
        if (color[v] == WHITE)  
            dfs(G, v)  
    color[u] ← BLACK  
}
```

dfs( G )



# DFS Trees

```

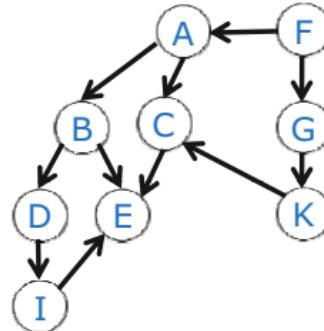
dfs( G=(V,E) )
for each v ∈ V
    color[v] ← WHITE
    p[v] ← null
for each v ∈ V
    if (color[v] == WHITE)
        dfs(G, v)
}

```

```

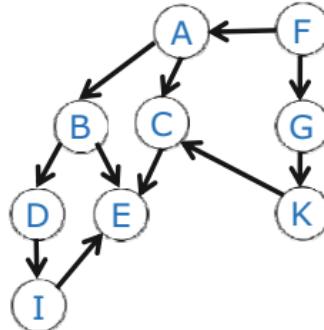
dfs( G=(V,E) , u)
color[u] ← GRAY;
for each v ∈ adj(u)
    if (color[v] == WHITE)
        p[v] ← u
        dfs(G, v)
color[u] ← BLACK
}

```



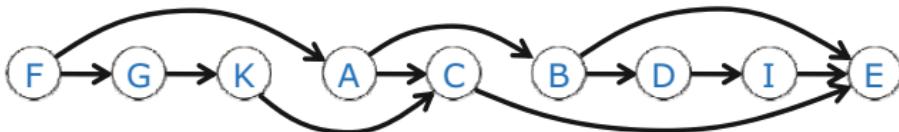
ได้ DFS tree ส่องด้น

# Topological Sort

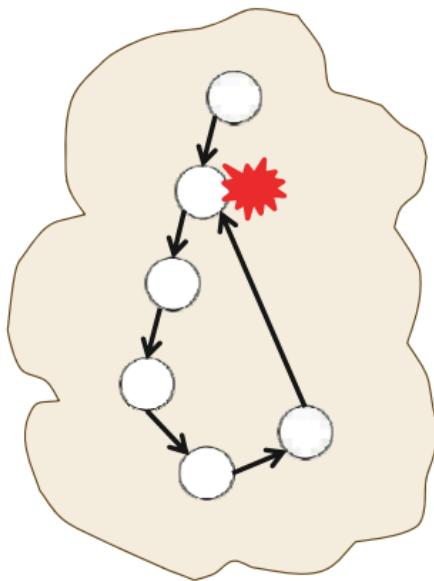


หา topological sort  
ได้เฉพาะกับ  
Directed Acyclic  
Graph (DAG)

ลำดับของปมที่ไม่มีเส้นเชื่อมซึ่งจากปมหลังมา�ังปมก่อนหน้า



# DAG : Directed Acyclic Graph

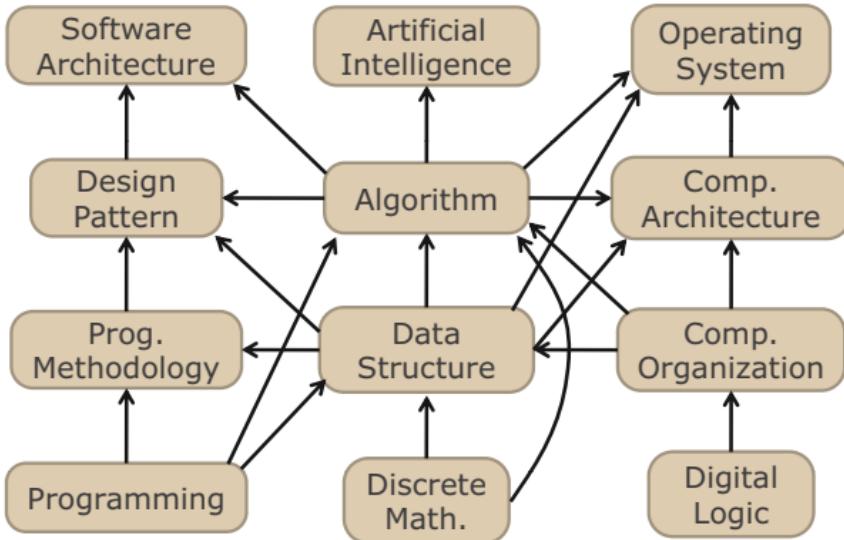


ปมเทา คือปมที่ยังไม่  
ผ่านลูกหลานไม่ครบ  
ดังนั้นถ้าพบปมเทา  
แสดงว่า พบรูปที่เป็น  
บรรพบุรุษ  
มี cycle ไม่ใช่ DAG

# isDAG : DFS

```
isDAG( G=(V,E) ) {  
    for each v ∈ V)  
        color[v] ← WHITE  
    for each v ∈ V)  
        if (color[v] == WHITE)  
            if ( !isDAG(G, v) ) return false  
    return true  
}  
isDAG( G=(V,E) , u ) {  
    color[u] ← GRAY  
    for each v ∈ adj(u)  
        if (color[v] == GRAY) return false  
        if (color[v] == WHITE)  
            if ( !isDAG(G, v) ) return false  
    color[u] ← BLACK  
    return true  
}
```

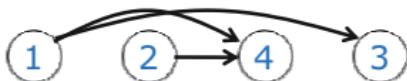
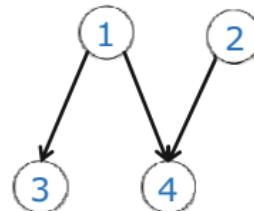
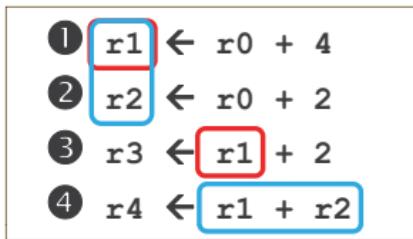
# Applications



# Applications

## Instruction scheduling

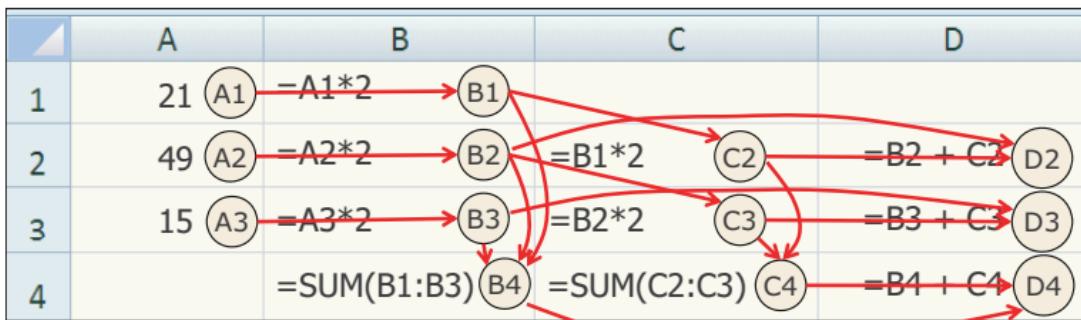
หน้าที่ของ compiler ในการจัดลำดับการทำงานของคำสั่ง เพื่อเพิ่มประสิทธิภาพการทำงานใน pipeline



# Applications

## Spreadsheet

ลำดับการคำนวณสูตรในเซลล์ต่าง ๆ เมื่อข้อมูลมีการเปลี่ยนแปลง



# DFS : เวลาเข้า-เวลาออก

```

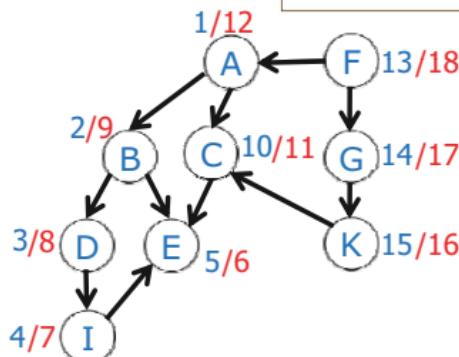
dfs( G=(V,E) ) {
    for each v ∈ V
        color[v] ← WHITE
    time ← 0
    for each v ∈ V
        if (color[v] == WHITE)
            dfs(G, v)
}

```

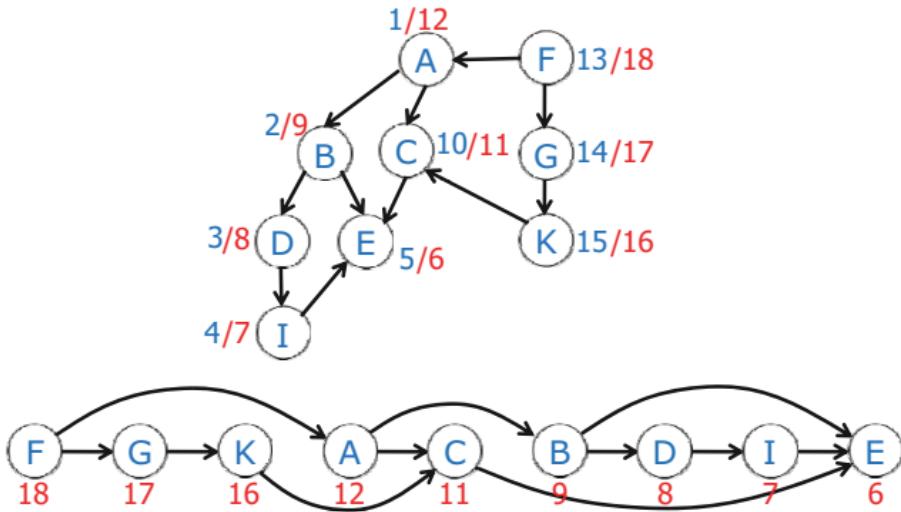
```

dfs(G=(V,E), u) {
    color[u] ← GRAY
    d[u] ← ++time;
    for each v ∈ adj(u)
        if (color[v] == WHITE)
            dfs(G, v)
    color[u] ← BLACK
    f[u] ← ++time;
}

```



# Topological Sort : DFS



ทำ DFS  
ลำดับปมจากซ้ายไปขวา  
จะเรียงตาม **f** จำนวนมากไปน้อย

## $(u, v) : f(u) > f(v)$



ไม่เกิด เพราะเป็น DAG



dfs( $G, u$ ) จะเสร็จได้  
 $f[u] \leftarrow ++time$

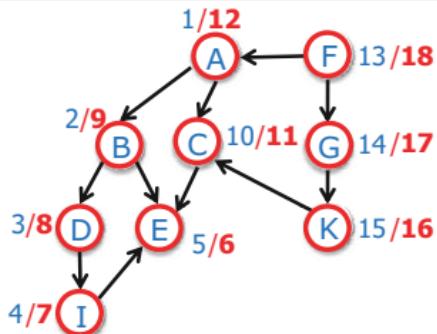
dfs( $G, v$ ) ต้องเสร็จก่อน  
 $f[v] \leftarrow ++time$



$f[u] \leftarrow ++time$

$f[v]$  ถูกใส่ค่ามาก่อนแล้ว

# Topological Sort : DFS



- topological sort คือลำดับของปมเรียงตาม  $f$  จากมากไปน้อย
- ใช้ stack จำ ปมที่ไปหาลูก ๆ เสร็จแล้ว
  - เมื่อ finish ก็ push ลง stack
- DFS เสร็จ : topological sort คือลำดับของปมที่ pop ได้จาก stack

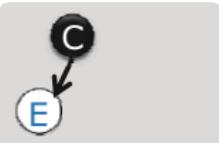
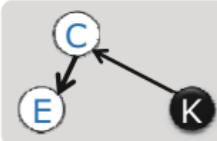
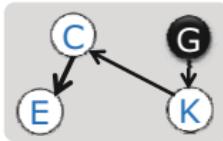
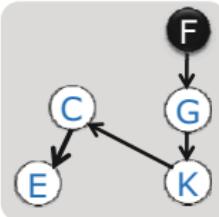
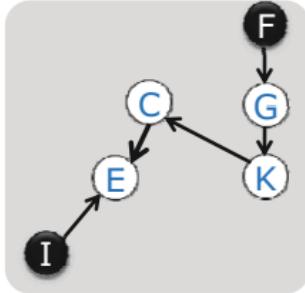
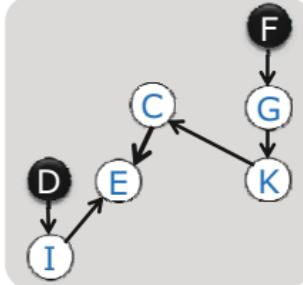
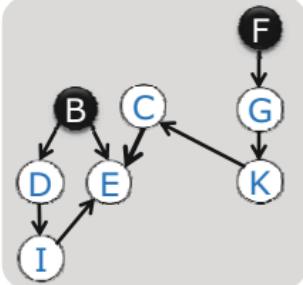
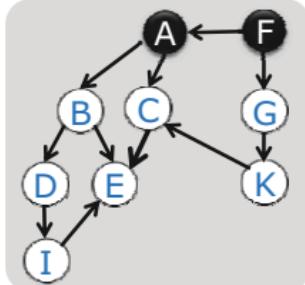
```

tsort( G = (V,E) ) {
    for each v ∈ V(G)
        color[v] ← WHITE
    S ← an empty stack
    for each v ∈ V(G)
        if (color[v] == WHITE)
            tsort( G, v )
    return S
}
  
```

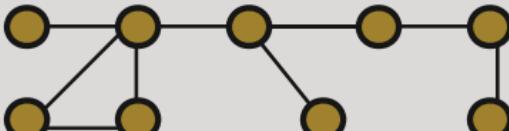
```

tsort( G=(V,E) , u ) {
    color[u] ← GRAY
    d[u] ← ++time
    for each v ∈ adj(u)
        if (color[v] == WHITE)
            tsort( G, v )
    color[u] ← BLACK;
    push(S, u)
}
  
```

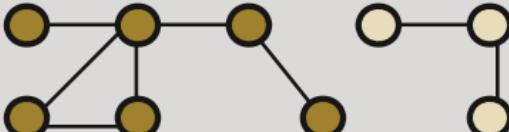
## Topological Sort : Kahn



# Undirected Graph : Connectivity



กราฟต่อถึงกันมี  
1 component



2 components

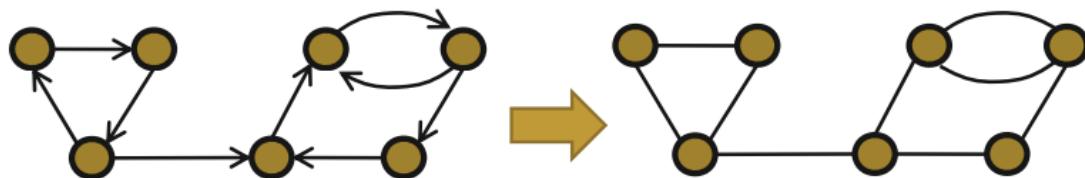
DFS ( G )

จำนวน components เท่ากับจำนวน DFS trees  
ปมในแต่ละ DFS tree ก็คือปมใน component เดียวกัน

# Directed Graph : Connectivity

## Weakly Connected

digraph ที่แปลงเป็นแบบไม่คิดทิศ แล้วเป็นกราฟต่อถึงกัน



# Directed Graph : Connectivity

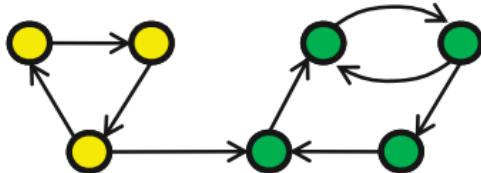
## Strongly Connected

กราฟที่มี path ระหว่างทุก ๆ คู่ปม ทั้งไปและกลับ

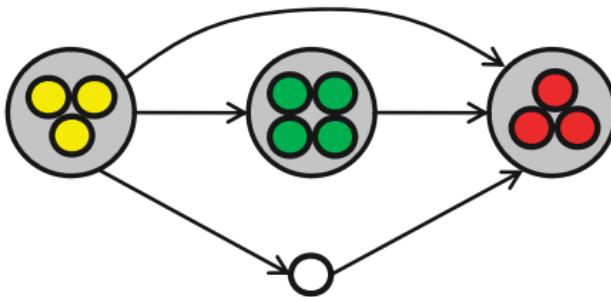
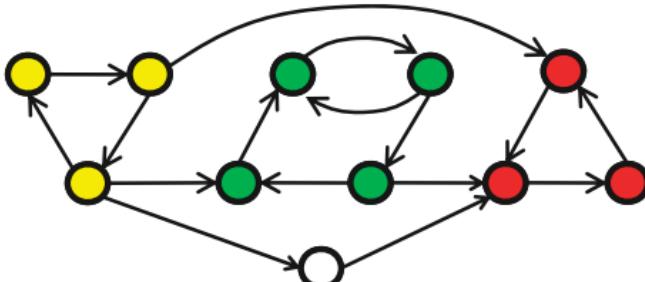


## Strongly Connected Components : SCC

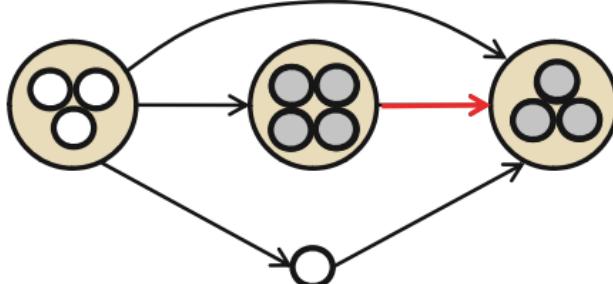
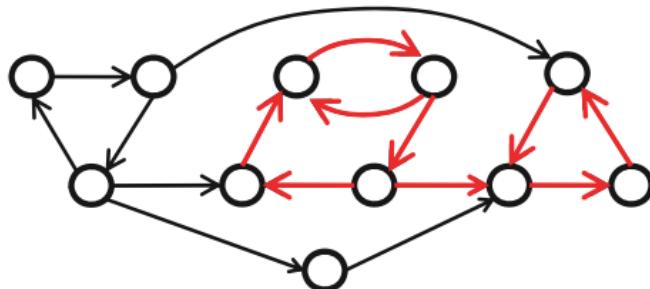
กราฟย่อยใหญ่สุด ๆ ที่ strongly connected



## Digraph = DAG ของ SCC

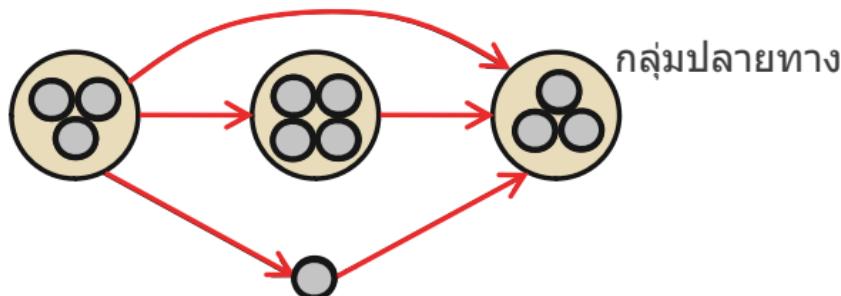


## DFS( $G, v$ ) : nodes ที่ไปถึงได้จาก $v$



## หา SCC : DFS(G, v)

ทำ DFS เริ่มที่ node ในกลุ่มปลายทางก่อน  
แล้วค่อยถอยมา DFS ที่กลุ่มต้นทาง

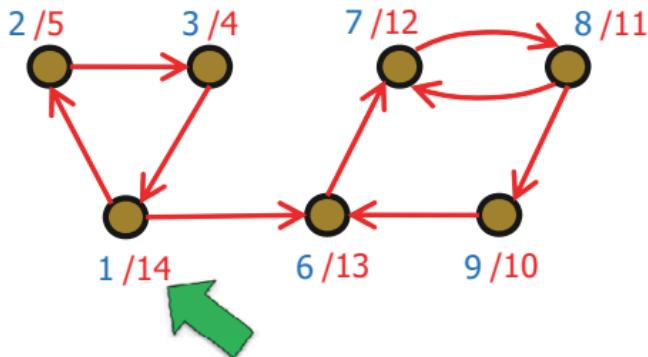


กลุ่มปลายทาง  
อยู่ไหน ?



หา topological sort  
ไม่ได้ เพราะไม่ใช่  
DAG

# กลุ่มต้นทาง หาจ่าวยกวา

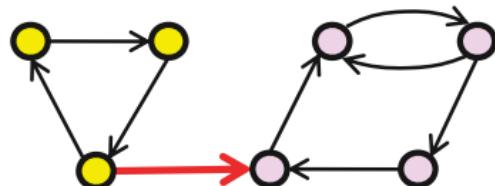


ทำ  $\text{DFS}(G)$  ปั้นที่มี  $f$  มากสุด อยู่ในกลุ่มต้นทาง

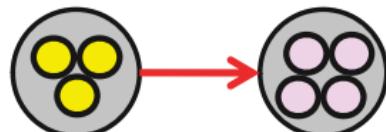
แต่เราต้องการกลุ่มปลายทาง!

# ปลายทางของ $G$ คือต้นทางของ $G^T$

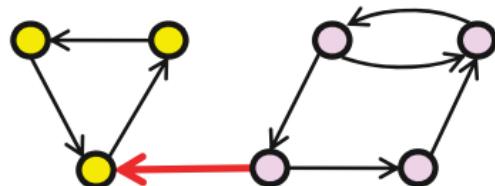
$G^T$  เมื่อนับ  $G$  แต่กลับทิศเส้นเชื่อม



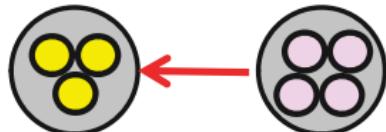
$G$



ปลายทาง



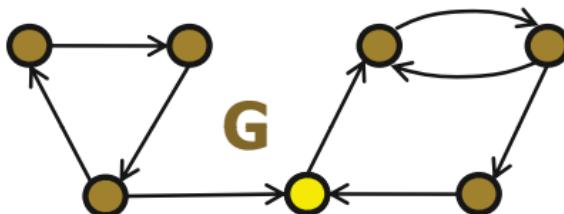
$G^T$



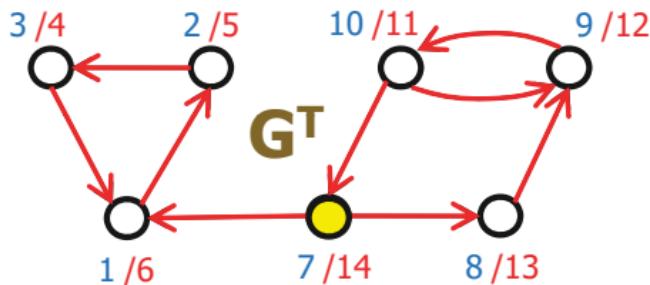
ต้นทาง

# ปลายทางของ $G$ คือต้นทางของ $G^T$

$G^T$  เมื่อน  $G$  แต่กลับทิศเส้นเชื่อม

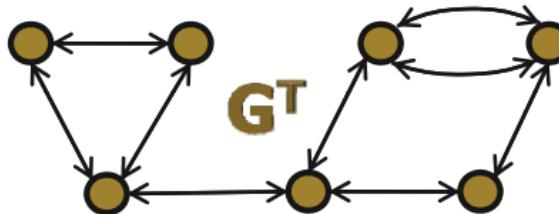


ทำ DFS(  $G^T$  ) ปมที่มี  $f$  มากสุด อยู่ในกลุ่มต้นทางของ  $G^T$   
คือกลุ่มปลายทางของ  $G$



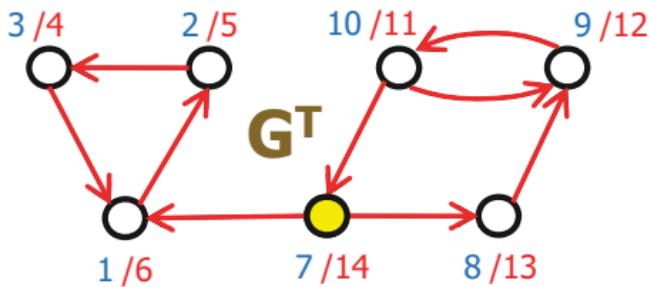
# วิธีหา SCC : Kosaraju's alg.

## 1. สร้าง $G^T$ (เหมือน $G$ แต่กลับทิศเส้นเชื่อม)



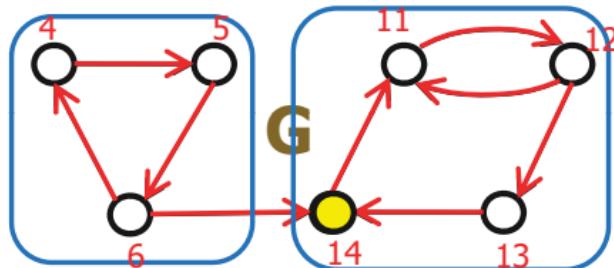
# วิธีหา SCC : Kosaraju's alg.

- สร้าง  $G^T$  (เนื่อง  $G$  แต่กลับทิศเส้นเชือม)
- $DFS(G^T)$  แล้วจำค่า  $f$  ระหว่างการผ่านปม



## วิธีหา SCC : Kosaraju's alg.

- สร้าง  $G^T$  ( เมื่อน  $G$  แต่กับทิศเส้นเชื่อม )
- DFS(  $G^T$  ) และจำค่า  $f$  ระหว่างการผ่านปม
- DFS(  $G$  ) ตามลำดับ  $f$  จากมากมาน้อย
- DFS tree ที่ได้แต่ละต้นคือ SCC


$$\Theta(|V| + |E|)$$

# อัลกอริทึมบนกราฟ

สมชาย ประสิทธิ์ยุตระกูต

# Graph Algorithms

## ❖ Minimum Spanning Trees

- ❖ Kruskal's algorithm
- ❖ Prim's algorithm

## ❖ Single-Source Shortest Paths

- ❖ Dijkstra's algorithm
- ❖ Bellman-Ford 's algorithm

## ❖ All-Pair Shortest Paths

- ❖ Floyd-Warshall's algorithm
- ❖ Johnson's algorithm

# Minimum Spanning Trees

## ❖ Input

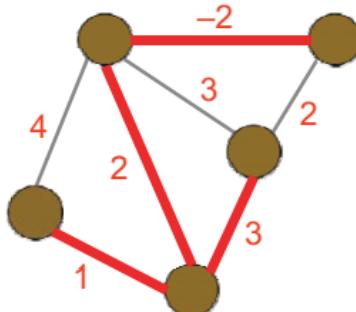
- ❖ กราฟ  $G = (V, E)$
- ❖ เส้นเชื่อมมีน้ำหนัก ไม่ต้องมีทิศ

## ❖ Output

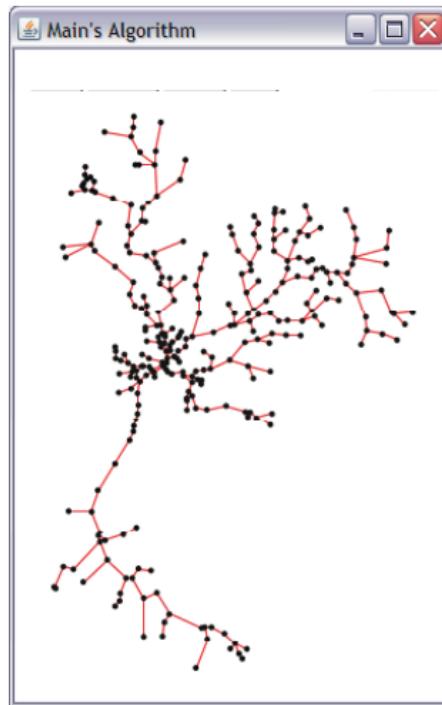
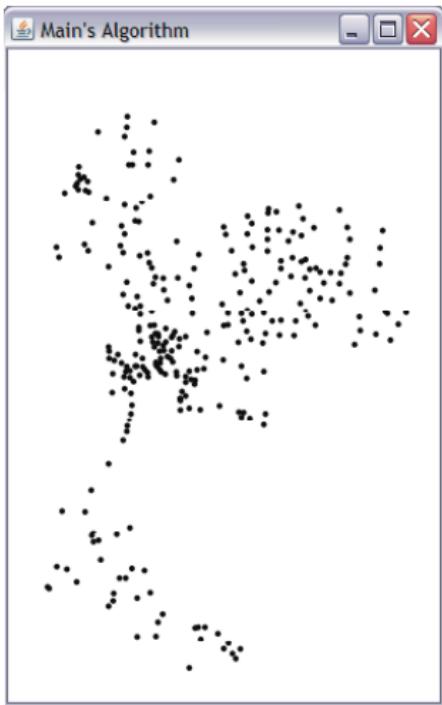
- ❖ เส้นเชื่อมต่าง ๆ (เป็นตันไม้) ที่ต่อถึงทุกปม
- ❖ มีผลรวมของน้ำหนักเส้นเชื่อมต่ำสุด

Spanning Tree

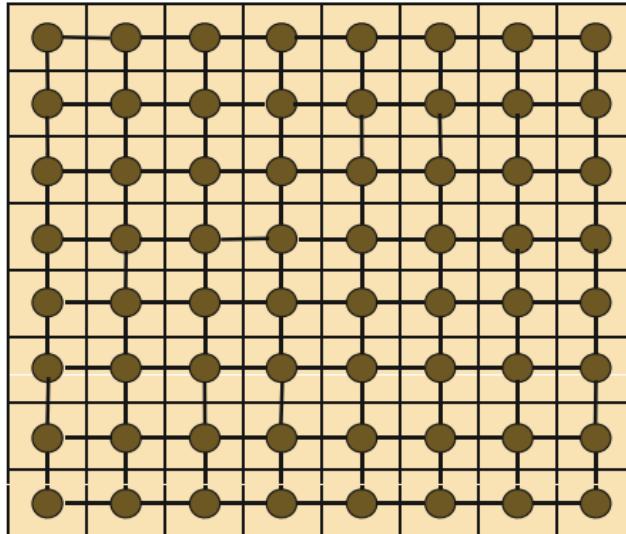
Minimum  
Spanning Tree



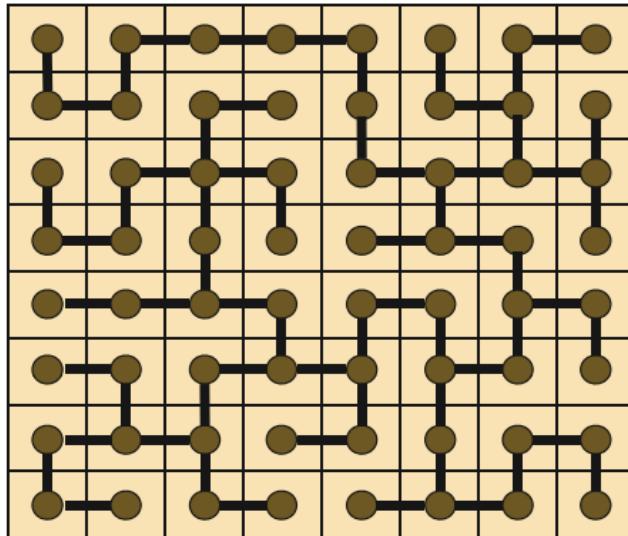
# ตัวอย่าง



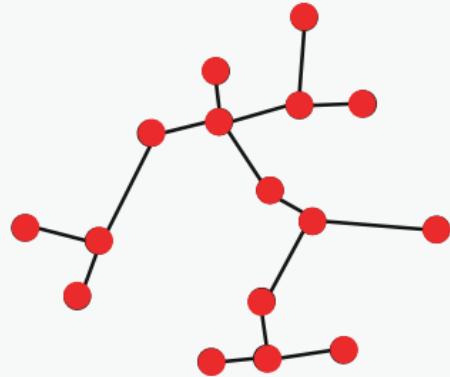
# สร้างเขาวงกตด้วย MST



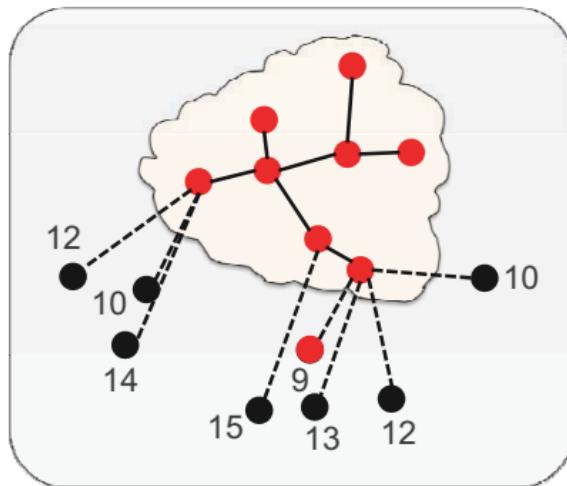
# สร้างเขาวงกตด้วย MST



# Prim's Algorithm



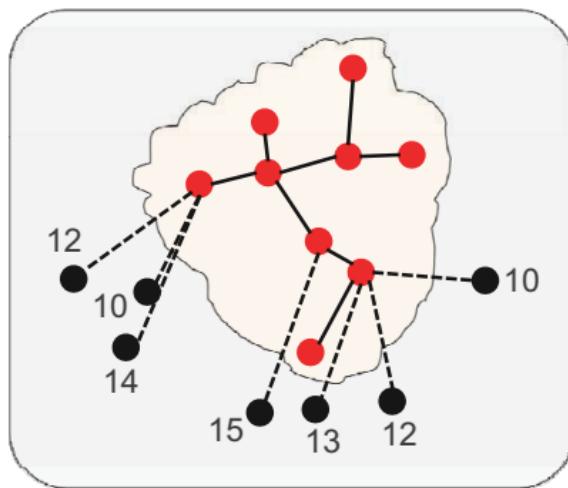
# Prim's Algorithm : Min-Heap



Prim is "greedy"

เลือกเส้นสั้นสุดที่ต่อ  
กับต้นไม้ที่กำลังโต

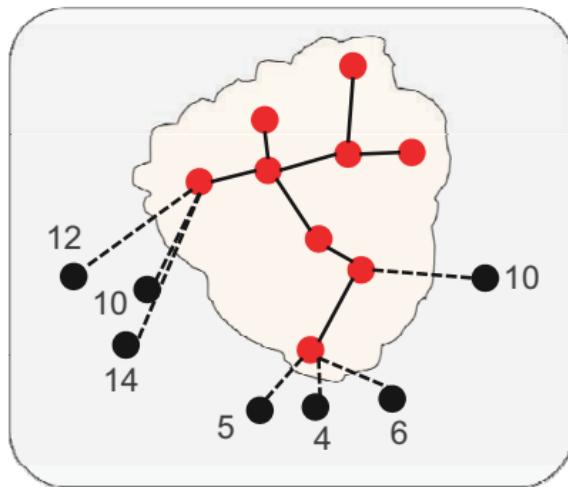
# Prim's Algorithm : Min-Heap



Prim is "greedy"

เลือกเส้นสั้นสุดที่ต่อ  
กับตันไม้ที่กำลังโต

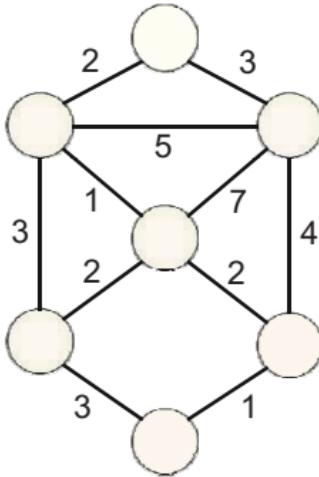
# Prim's Algorithm : Min-Heap



Prim is "greedy"

เลือกเส้นลั่นสุดที่ต่อ  
กับตันไม้ที่กำลังโต

## ตัวอย่าง : Prim



# Prim's Algorithm

```
Prim( G=(V,E) ) {
    for each vertex v in V {
        d[v] = ∞; p[v] = null
        inMST[v] = false
    }
    select an arbitrary vertex v and let's d[v] = 0
    H = a min heap of all vertices ordered by d[]
    while( H ≠ ∅ ) {
        u = H.removeMin()
        inMST[u] = true
        for each v ∈ adj(u) {
            if( !inMST[v] AND w(u,v) < d[v] ) {
                d[v] = w(u,v); H.decreaseKey(v)
                p[v] = u
            }
        }
    }
    return p;
}
```

ใช้ binary heap

dense graph →  $e = \Theta(v^2)$

$O(e \log v)$

$O(v^2 \log v)$

# Prim's Algorithm

```

Prim( G[1..n] [1..n] ) {
    for ( v = 1; v <= n; v++ ) {
        d[v] =  $\infty$ ; p[v] = null
        inMST[v] = false
    }
    select an arbitrary vertex v and let's d[v] = 0
    for ( i = 1; i <= n; i++ ) {
        u = minIndex( d ); d[u] =  $\infty$  ←  $\Theta(n)$ 
        inMST[u] = true
        for ( v = 1; v <= n; i++ ) {
            if( !inMST[v] AND G[u][v] < d[v] ) {
                d[v] = G[u][v];
                p[v] = u
            }
        }
    }
    return p;
}

```

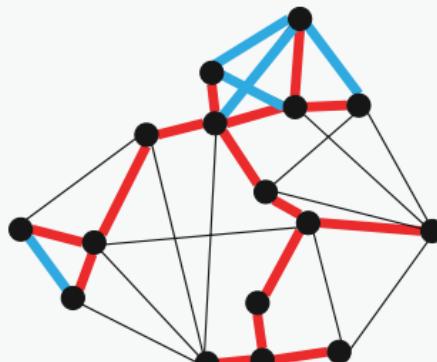
ใช้ adjacency matrix

$\Theta(n^2)$

$\Theta(v^2)$

# Kruskal's Algorithm

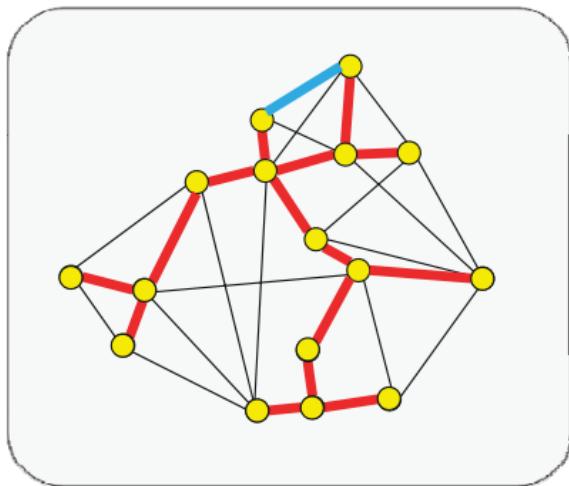
ตรวจว่าเกิดวงอย่างไร (เร็ว ๆ)



Kruskal is "greedy"

เลือกเส้นสั้นสุดที่เชื่อมตันไม่  
สองตันที่ต่างกันในป่า

# Kruskal's Algorithm



Kruskal is "greedy"

เลือกเส้นสั้นสุดที่เชื่อมตันไม้  
สองตันที่ต่างกันในป่า

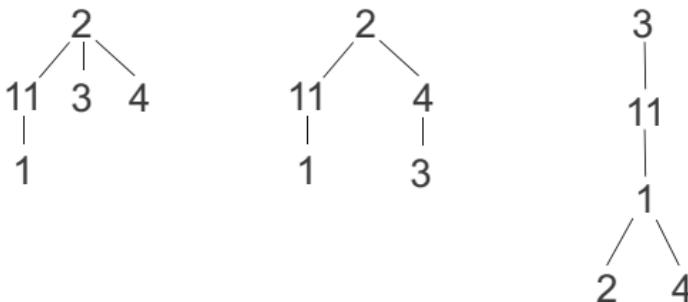
# Kruskal's Algorithm

```
Kruskal( G=(V,E) ) {
    for each v in V  create a set {v}

    H = a min heap of all edges ordered by weights
    T = an empty list
    while ( T.size() < |V|-1 ) {
        (u,v) = H.removeMin()
        if ( findSet(u) ≠ findSet(v) ) {
            T.add( (u,v) )
            unionSet( findSet(u) , findSet(v) )
        }
    }
    return T
}
```

# แทนเซตด้วยต้นไม้

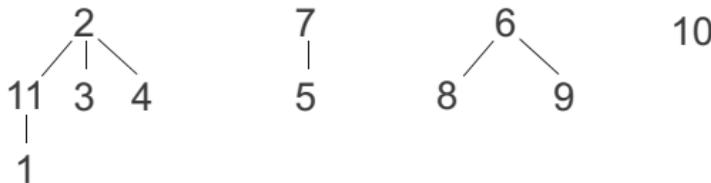
{ 1, 2, 3, 4, 11 }



แบบใดก็ได้ ออยู่ในต้นเดียวกัน เป็นเซตเดียวกัน

# แทนกลุ่มเซตด้วยป่าไม้

$\{11, 1, 2, 3, 4\}, \{5, 7\}, \{6, 8, 9\}, \{10\}$



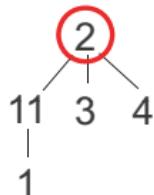
กลุ่มเซตไม่มีส่วนร่วม

(Disjoint Sets)

ขอใช้กลุ่มเซตที่สมาชิกเป็นจำนวนเต็ม  
 $1, 2, \dots, n$

# findSet และ unionSet

ให้รากแทนหมายเลขเชด



$\text{findSet}(1)$  ได้ 2

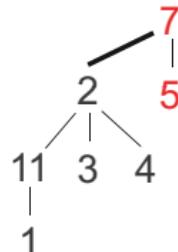
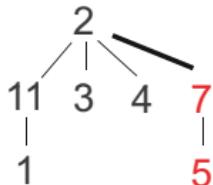
$\text{findSet}(3)$  ได้ 2

...

$\text{findSet}(5)$  ได้ 7

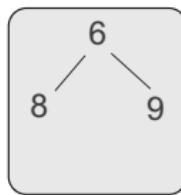
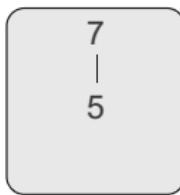
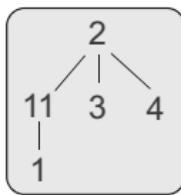
$\text{findSet}(7)$  ได้ 7

$\text{unionSet}(2, 7)$



## การแทน Disjoint Sets

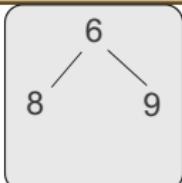
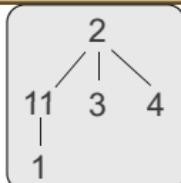
{ 11, 1, 2, 3, 4 }, { 5, 7 }, { 6, 8, 9 }, { 10 }



	1	2	3	4	5	6	7	8	9	10	11
P	11	2	2	2	7	6	7	6	6	10	2

$P[k]$  คือหมายเลขพ่อของ  $k$

# Disjoint Sets : findSet



	1	2	3	4	5	6	7	8	9	10	11
P	11	2	2	2	7	6	7	6	6	10	2

```

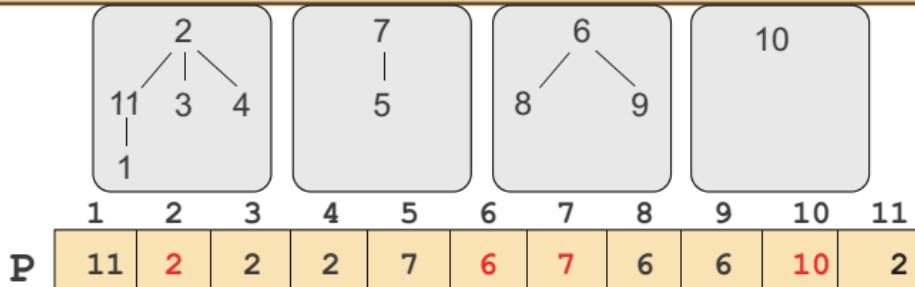
findSet( e ) {
    while( P[e] != e ) {
        e = P[e]
    }
    return e
}
  
```

```

findSet( e ) {
    if ( P[e] == e )
        return e
    else
        return findSet(P[e])
}
  
```

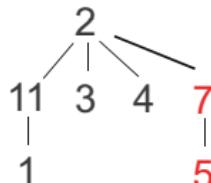
ใช้เวลาแปรตามความสูงของต้นไม้

# Disjoint Sets : unionSet

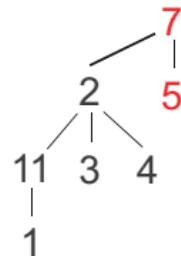


```
unionSet( s, t ) {  
    P[t] = s;  
}
```

unionSet(2, 7)

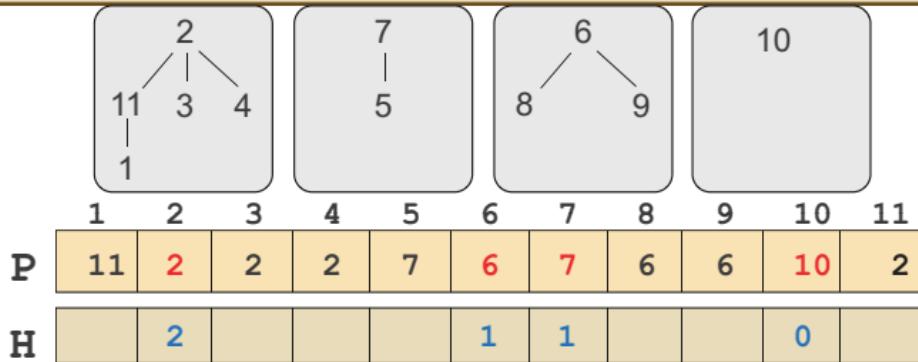


unionSet(7, 2)



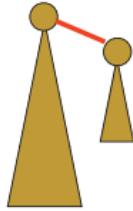
ต้องการตันไม้เดี้ย  
ทำให้ findSet เร็ว

# Disjoint Sets : unionSet



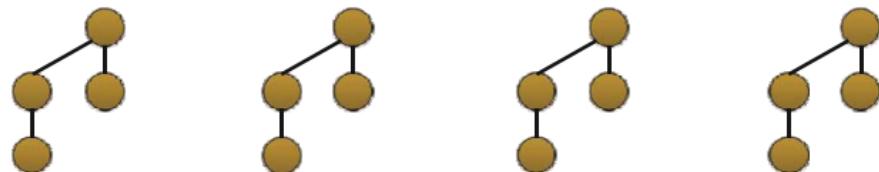
```
unionSet( s, t ) {
    if ( H[s] > H[t] ) {
        P[t] = s;
    } else {
        P[s] = t;
        if (H[s] == H[t]) H[t]++;
    }
}
```

O(1)

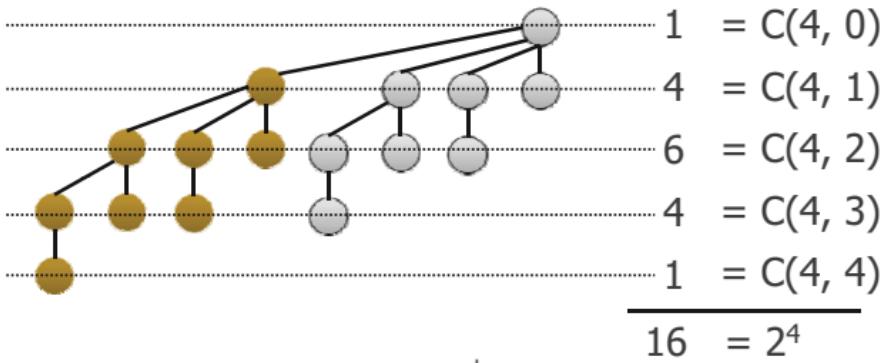


ต้นเดียเป็นลูกของ  
รากของต้นสูง

# unionSet ให้ต้นไม้สูงสุด ๑



# unionSet ให้ต้นไม้สูงสุด ๑



unionSet จนข้อมูล  $n$  ตัวอยู่ในหนึ่งเขต

จะได้ต้นไม้สูงสุด สูง  $\log_2 n$

$$\sum_{k=0}^n C(n,k) = 2^n$$

สรุป : unionSet  $\rightarrow O(1)$

findSet  $\rightarrow O(\log n)$

# Kruskal's Algorithm

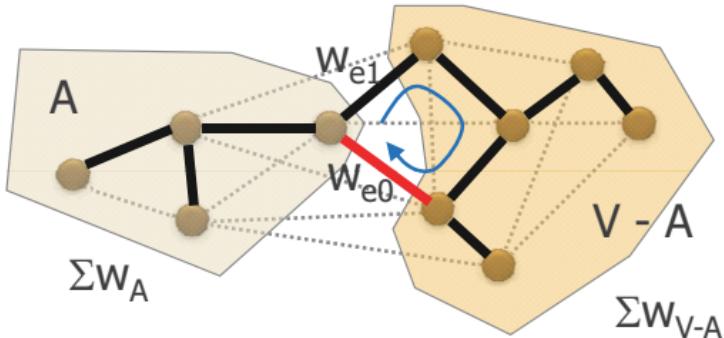
```
Kruskal( G=(V,E) ) {  
    D = a new group of disjoint sets  
    for each v in V  
        D.createNewSet(v)  
    H = a min heap of all edges ordered by weights  
    T = an empty list  
    while ( T.size() < |V|-1 ) {  
        (u,v) = H.removeMin()  
        if ( D.findSet(u) ≠ D.findSet(v) ) {  
            T.add( (u,v) )  
            D.unionSet( D.findSet(u) , D.findSet(v) )  
        }  
    }  
    return T  
}
```

$O(e \log e)$

simple graphs :  $e = O(v^2)$

$O(e \log v)$

# เลือกสันสุดนำไปสู่ MST



ให้ต้นดำเป็น MST

ดังนั้น  $\Sigma W_A + w_{e1} + \Sigma W_{V-A}$  ต่ำสุด

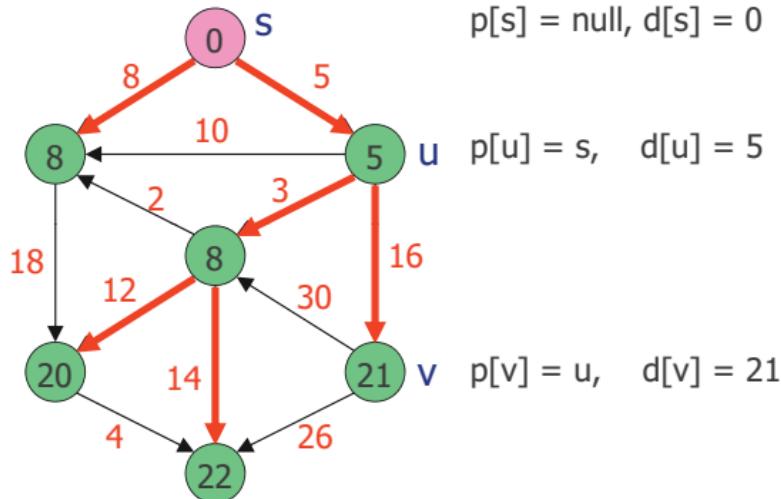
ให้  $w_{e0} \leq w_{e1}$

ลบ  $e_1$ , เพิ่ม  $e_0$

ได้ spanning tree ใหม่ที่มี  $\Sigma w \leq \Sigma w$  ของ MST  
แสดงว่า  $e_0$  อยู่ใน MST

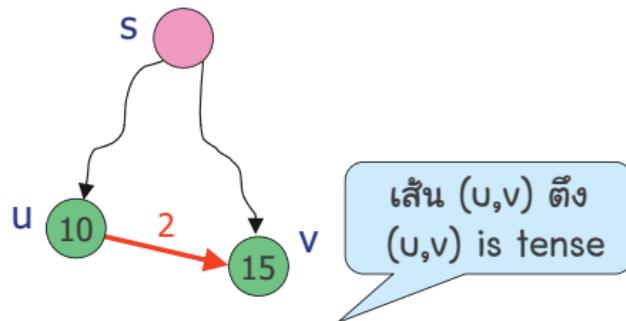
# วิถีสั้นสุดจากแหล่งต้นทางเดียว

- ให้  $G$  เป็นกราฟระบุทิศทาง เส้นเชื่อมมีความยาว
- ต้องการหาวิถีสั้นสุดจากปม  $s$  ใน  $G$  ไปยังปมอื่น ๆ
- สิ่งที่ได้ : Shortest-path tree



# เส้นตึง + คลายเส้น

$d[u]$  เก็บความยาวของวิถีสั้นสุดเท่าที่รู้จาก  $s$  ไป  $v$



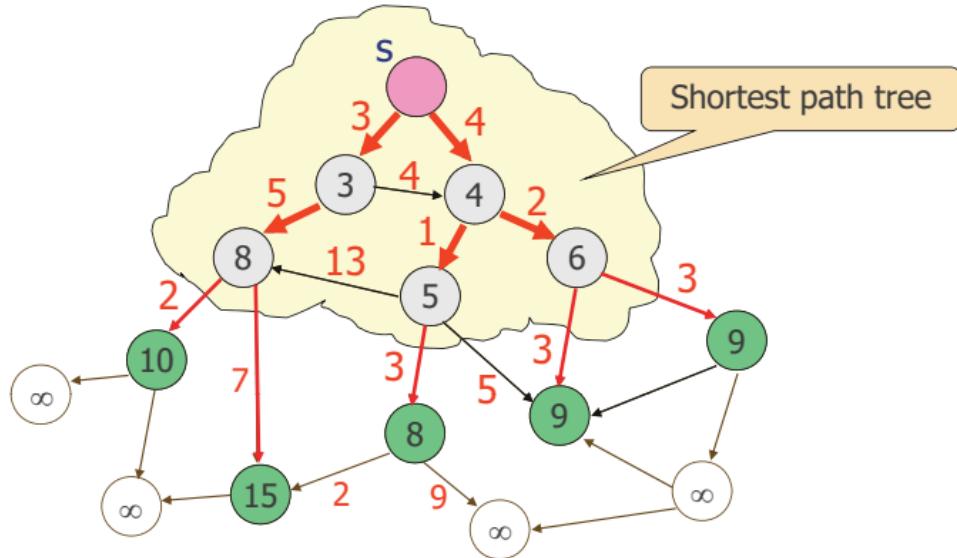
พิจารณาเส้นเชื่อม  $(u,v)$  พนว่า  $d[u] + w(u,v) < d[v]$

แสดงว่า พบริถีใหม่ผ่าน  $(u,v)$  ไป  $v$  ที่สั้นกว่าของเดิมที่เป็น  $d[v]$

ดังนั้นเปลี่ยน  $d[v] = d[u] + w(u,v)$  และเปลี่ยน  $p[v] = u$

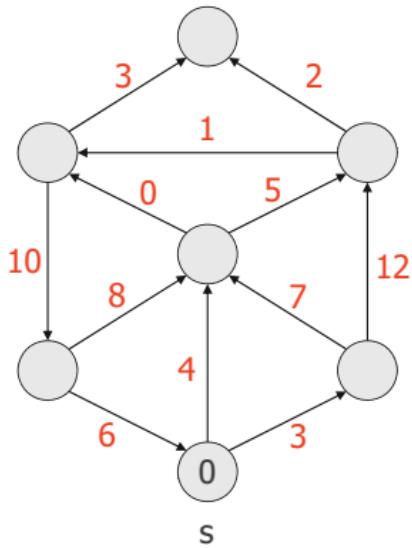
คลายเส้น  $(u,v)$   
relax  $(u,v)$

# แนวคิดของ Dijkstra's Algorithm



ขยาย Shortest path tree  
โดยเลือกปมที่มี  $d[]$  ต่ำสุด

# Dijkstra Algorithm : ตัวอย่าง



# อัลกอริทึมของ Dijkstra

```
Dijkstra( G=(V,E) , s ) {  
    for each v ∈ V {  
        d[v] = ∞; p[v] = null;  
    }  
    d[s] = 0  
    H = a min heap of all vertices ordered by d[]  
    while (H ≠ Ø) {  
        u = H.removeMin()  
        for each v ∈ Adj(u) {  
            if ( d[u]+w(u,v) < d[v] ) {  
                d[v] = d[u] + w(u,v); H.decreaseKey(v)  
                p[v] = u  
            }  
        }  
    }  
    return d  
}
```

$O(e \log v)$

ถ้า เส้น  $(u,v)$  ตีง

relax  $(u,v)$

แต่ละปมถูกนำออกจากชีปหนึ่งครั้ง  
เส้นเชื่อมแต่ละเส้นถูกคลายอย่างมากหนึ่งครั้ง

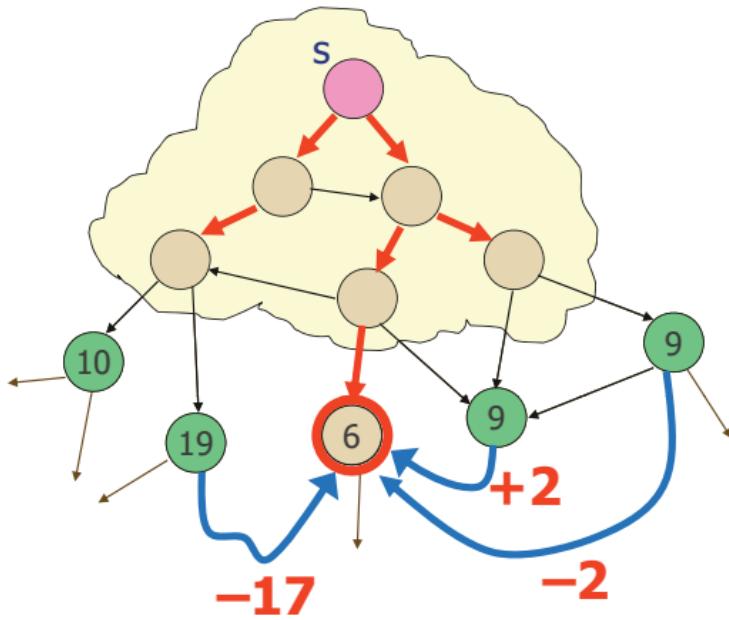
# Dijkstra คล้าย Prim

```

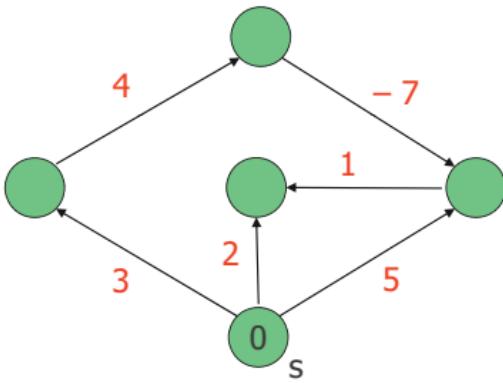
Dijkstra( G=(V,E) , s ) {
    for each v in V {
        d[v] =  $\infty$ ; p[v] = null
        inMST[v] = false
    }
    select an arbitrary vertex v and let's d[v] = 0
    H = a min heap of all vertices ordered by d[]
    while( H  $\neq \emptyset$  ) {
        u = H.removeMin()
        inMST[u] = true
        for each v  $\in$  adj(u) { d[u] +
            if( inMST[v] AND w(u,v) < d[v] ) {
                d[v] = w(u,v); H.decreaseKey(v)
                p[v] = u
            }
        }
    }
    return p;
}

```

## ถ้ามี negative-weight edges

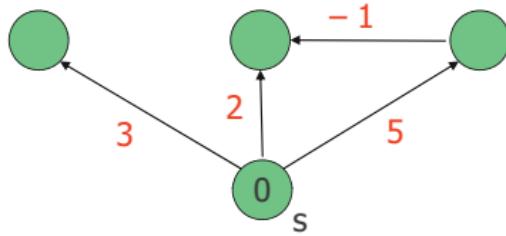


# ถ้ามี negative-weight edges



Dijkstra's algorithm  
อาจให้ค่าตอบผิด

# ถ้ามี negative-weight edges



Dijkstra's algorithm  
อาจให้ค่าตอบถูก

# ลองคิดดู

```

ModifiedDijkstra( G=(V,E) , s ) {
    for each v ∈ V {
        d[v] = ∞; p[v] = null;
    }
    d[s] = 0
    H = an empty min heap;
    H.add(s)
    while (H ≠ ∅) {
        u = H.removeMin()
        for each v ∈ Adj(u) {
            if ( d[u]+w(u,v) < d[v] ) {
                d[v] = d[u] + w(u,v); H.decreaseKey(v)
                p[v] = u
                H.add(v)
            }
        }
    }
    return d
}

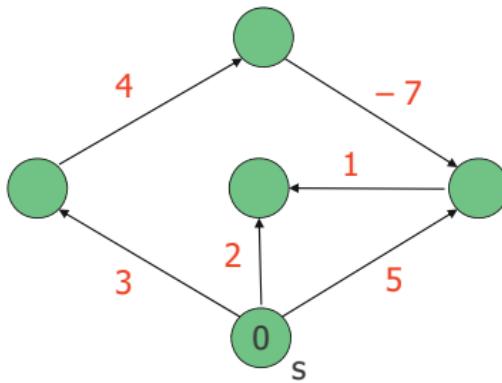
```

อัลกอริทึมนี้รองรับเล่นเชื่อม  
ความยาวลบได้หรือไม่ ?

มีข้อเสียอะไร ?

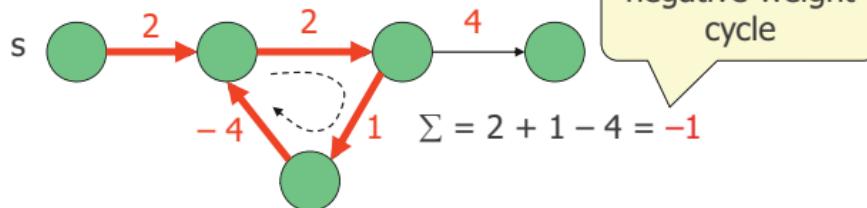
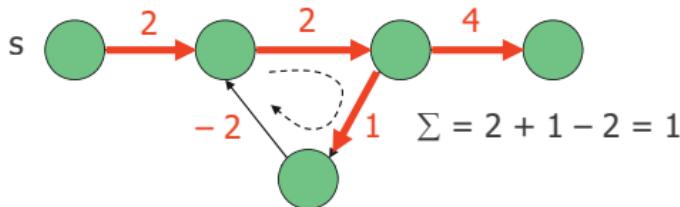
เพิ่มปมปลายเล่นเชื่อม  
ที่เคยตึง ให้กับ heap

# Modified Dijkstra



มีข้อเสียอะไร ?

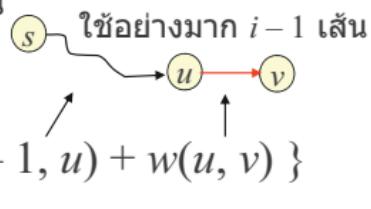
# Negative-Weight Cycles



หาวิถีสั้นสุดไม่ได้  
เพราะมันสั้นลงเรื่อย ๆ เมื่อหมุนในวง

# อัลกอริทึมของ Bellman-Ford

- ❖ ใช้ได้กับ negative-weight edge
- ❖ จะแจ้งให้ทราบ ถ้ามี negative-weight cycle
- ❖ ใช้ dynamic programming
- ❖ วิถีสั้นสุดจาก  $s$  ไป  $t$ 
  - ❖ ต้องไม่ผ่านปมซ้ำ
  - ❖ มีเส้นเชื่อมในวิถีอย่างมาก  $|V| - 1$  เส้น
- ❖ ให้  $d(i, v)$  แทนความยาวของวิถีสั้นสุดจาก  $s$  ไป  $v$  ที่มีเส้นเชื่อมอย่างมาก  $i$  เส้น

$$d(i, v) = \min \left\{ \begin{array}{l} d(i-1, v) \\ \min_{u \in \text{Adj}(v)} \{ d(i-1, u) + w(u, v) \} \end{array} \right.$$


$$d(?, s) = 0, d(0, v) = \infty \quad \text{ต้องการ } d(|V|-1, v)$$

# อัลกอริทึมของ Bellman-Ford

$$d_i(v) = \min \begin{cases} d_{i-1}(v) \\ \min_{u \in \text{Adj}(v)} \{ d_{i-1}(u) + w(u, v) \} \end{cases}$$

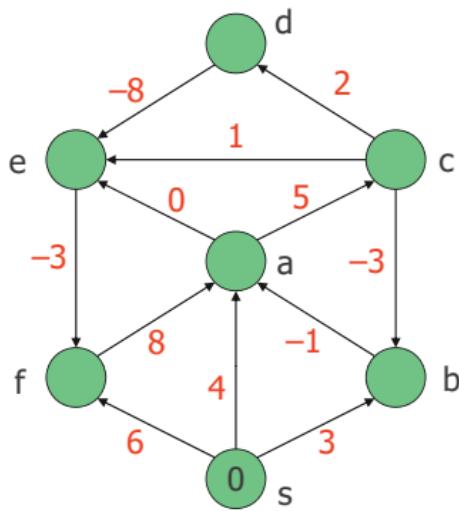
```

BellmanFord(G=(V,E) , s) {
    for each v ∈ V {
        d[v] = ∞; p[v] = null;
    }
    d[s] = 0
    for (i=1; i<|V|; i++) {
        for each edge (u,v) in E
            if ( d[u]+w(u,v) < d[v] ) {
                d[v] = d[u] + w(u,v)
                p[v] = u;
            }
    }
    return d
}

```

 $\Theta(|V|)$  $\Theta(|E|)$  $\Theta(v\epsilon)$

# Bellman-Ford : ตัวอย่าง



s, a, b, c, d, e, f

# มี negative-weight cycle ?

- ❖ relax เส้นตึงทุกเส้นเป็นจำนวน  $|V|-1$  รอบ
- ❖ ตรวจสอบเส้นเชื่อมทุกเส้นอีกรอบ
- ❖ ถ้ายังมีเส้นตึง  $\rightarrow$  มี negative-weight cycle

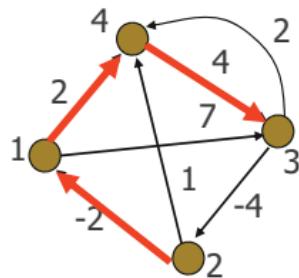
```

BellmanFord( G=(V,E) , s ) {
    for each v ∈ V {
        d[v] = ∞; p[v] = null;
    }
    d[s] = 0
    for (i=1; i<|V|; i++)
        for each edge (u,v) ∈ E
            if ( d[u]+w(u,v) < d[v] ) {
                d[v] = d[u] + w(u,v); p[v] = u;
            }
    for each edge (u,v) ∈ E
        if ( d[u]+w(u,v) < d[v] ) return null
    return d;
}

```

# วิถีสั้นสุดของทุกคู่ปมในกราฟ

กราฟ  $G(V, E)$  เส้นเชื่อมมีน้ำหนัก มีทิศทาง



เส้นเชื่อมยาวเป็นลบได้  
แต่ต้องไม่มีวงที่ความยาวรวมเป็นลบ

shortest path tree ของวิถีสั้น  
สุดจากปม 2 ถึงปมอื่น ๆ

$W$	1	2	3	4
1	0	-	7	2
2	-2	0	-	1
3	-	-4	0	2
4	-	-	4	0

$D$	1	2	3	4
1	0	2	6	2
2	-2	0	4	0
3	-6	-4	0	-4
4	-2	0	4	0

$P$	1	2	3	4
1	-	3	4	1
2	2	-	4	1
3	2	3	-	1
4	2	3	4	-

# การหาวิถีสั้นสุดของทุกคู่ปม

## ❖ หา single-source shortest paths ของทุกปม

- ❖ เรียก Dijkstra  $v$  ครั้ง (ความยาวเส้นเชื่อมห้ามเป็นลบ)
  - ❖ binary heap + adj. list :  $O(v \cdot e \log v)$
  - ❖ linear array + adj. matrix :  $O(v \cdot v^2) = O(v^3)$
- ❖ เรียก Bellman-Ford  $v$  ครั้ง (เส้นเชื่อมยาวเป็นลบได้)
  - ❖  $O(v^2e)$  ซึ่งเป็น  $O(v^4)$  สำหรับ dense graph

## ❖ ใช้กำหนดการพลวัต

- ❖ ตรงไปตรงมา :  $\Theta(v^4)$
- ❖ repeated squaring :  $\Theta(v^3 \log v)$
- ❖ Floyd-Warshall :  $\Theta(v^3)$

# นิยามความยาวของวิถีสั้นสุด

## ❖ Single-source

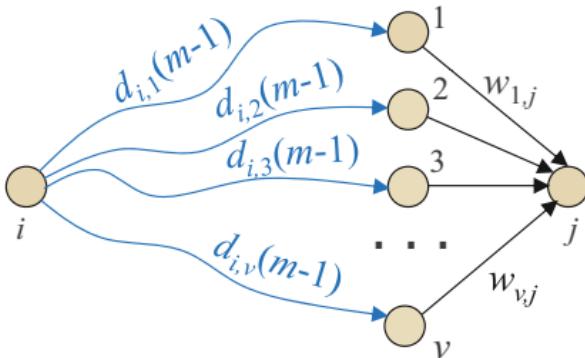
- ❖ ให้  $d_m(j)$  แทนความยาวของวิถีสั้นสุดจาก  $s$  ไป  $j$   
ที่มีเส้นเชื่อมอย่างมาก  $m$  เส้น

## ❖ All-pair

- ❖ ให้  $d_m(i, j)$  แทนความยาวของวิถีสั้นสุดจาก  $i$  ไป  $j$   
ที่มีเส้นเชื่อมอย่างมาก  $m$  เส้น
- ❖ ขอเปลี่ยนเป็น  $d_m(i, j)$  เป็น  $d_{i,j}(m)$

# คำตอบใหญ่ได้จากคำตอบย่อย

- ให้  $d_{i,j}(m)$  แทนความยาวของวิถีสั้นสุดจาก  $i$  ไป  $j$   
ที่ใช้เส้นเชื่อมไม่เกิน  $m$  เส้น

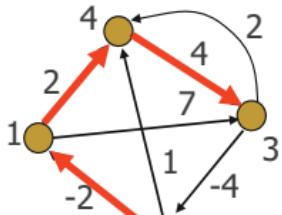


$$d_{i,j}(m) = \min_{1 \leq k \leq v} \{ d_{i,k}(m-1) + w_{k,j} \}$$

$$d_{i,i}(0) = 0, \quad d_{i,j}(0) = \infty, \quad d_{i,j}(1) = w_{i,j}, \quad w_{i,i} = 0$$

# การคำนวณความยาวของวิถีสั้นสุด

- ❖ วิถีสั้นสุดย่อมไม่ผ่านปมข้า
- ❖ ถ้า  $G$  มี  $v$  ปม, วิถีสั้นสุดของปมคู่ใด ๆ ต้องใช้เส้นเชื่อมไม่เกิน  $v - 1$  เส้น



ความยาวของวิถีสั้นสุด  
จาก  $i$  ไป  $j$  ที่ใช้เส้น  
เชื่อมไม่เกิน  $m$  เส้น

- ❖ ให้  $D(m)$  คือเมตริกซ์ที่เก็บ  $d_{ij}(m)$ 
  - ❖  $D(1) = W$  ( $W$  คือ adjacency matrix ของ  $G$ )
- ❖ เริ่มที่  $D(1)$  คำนวณหา  $D(2), D(3), \dots, D(v - 1)$

$$d_{i,j}(m) = \min_{1 \leq k \leq v} \{ d_{i,k}(m-1) + w_{k,j} \}$$

# แบบช้า

```
APSP_Slow( W[1..v][1..v] ) {
    D[1] = W
    for(m = 2; m < v; m++)
        D[m] = ExtendShortestPath( D[m-1], W );
    return D[v-1];
}
```

$\Theta(v^4)$

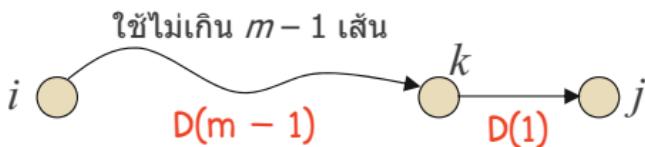
หา  $D(2), D(3), \dots, D(v - 1)$

```
ExtendShortestPath( d[1..v][1..v], W[1..v][1..v] ) {
    for (i = 1; i <= v; i++) {
        for (j = 1; j <= v; j++) {
            dm[i][j] =  $\infty$ ;
            for (k = 1 to v) {
                dm[i][j] = min( dm[i][j], d[i][k] + w[k][j] );
            }
        }
    }
    return dm;
}
```

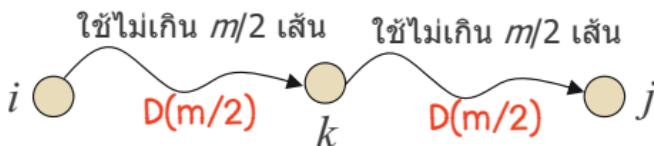
$d_{i,j}(m) = \min_{1 \leq k \leq v} \{ d_{i,k}(m-1) + w_{k,j} \}$

# ปรับปรุงให้เร็วขึ้น

- ❖ แบบช้า :  $D(m)$  หาจาก  $D(m-1)$  กับ  $D(1)$



- ❖ แบบเร็ว :  $D(m)$  หาจาก  $D(m/2)$  กับ  $D(m/2)$



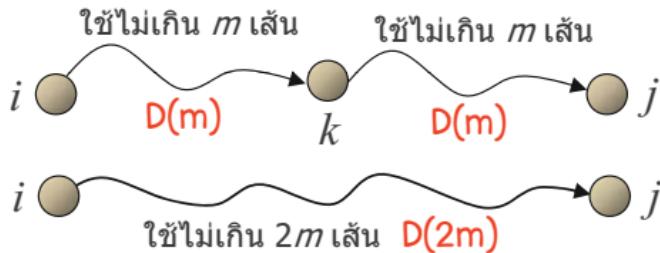
- ❖ เริ่ม  $D(1)$  หา  $D(2), D(4), D(8), \dots$  (เร็วกว่า)
- ❖ กราฟไม่มีวงลบ,  $D(v-1)=D(v)=D(v+1)=\dots$
- ❖ ต้องการ  $D(v-1)$  ก็หา  $D(2^k)$   $2^k \geq v-1$ 
  - ❖ เช่น ต้องการ  $D(13)$  ก็หา  $D(16)$ , ต้องการ  $D(57)$  ก็หา  $D(64)$

# แบบกำลังสองช้า

```

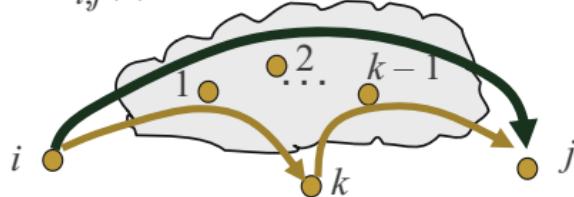
APSP_RepeatedSquare( W[1..v] [1..v] ) {
    D      = W;
    m = 1;
    while (m < v-1) {
        D..... = ExtendShortestPath( D....., D..... );
        m = 2*m;
    }
    return D ;
}
    
```

$2^c \geq v-1$   
 $c \geq \log_2(v-1)$   
 $c = \lceil \log_2(v-1) \rceil$



# อัลกอริทึมของ Floyd-Warshall

- ❖ เปลี่ยนแนวคิดการเขียน recurrence
- ❖ ให้  $d_{i,j}(k)$  แทนความยาวของวิถีสั้นสุดจากปม  $i$  ไป  $j$  ที่สามารถใช้จุดใน  $\{1,2,3,\dots,k\}$  เป็นปมระหว่างทาง
  - ❖  $d_{i,j}(0)$  คือ  $w_{i,j}$  ("ไม่มีปมระหว่างทาง")
  - ❖ ต้องการ  $d_{i,j}(v)$ ,  $v = \text{จำนวนปม}$  (สามารถใช้ได้ทุกปม)



$$d_{i,j}(k) = \min( d_{i,j}(k-1), d_{i,k}(k-1) + d_{k,j}(k-1) )$$

$$d_{i,j}(0) = w_{i,j}$$

# อัลกอริทึมของ Floyd-Warshall

```

FloydWarshall( W[1..v] [1..v] ) {
    D = W
    for (k = 1; k <= v; k++) {
        for (i = 1; i <= v; i++) {
            for (j = 1; j <= v; j++) {
                D[i][j] = min( D[i][j],
                                D[i][k] + D[k][j] );
            }
        }
    }
    return D;
}

```

ให้เข้าม {1,2,...,k}  
เป็นปมระหว่างทาง

$\Theta(v^3)$

$$\begin{aligned}
 d_{i,j}(k) &= \min( d_{i,j}(k-1), d_{i,k}(k-1) + d_{k,j}(k-1) ) \\
 d_{i,j}(0) &= w_{i,j}
 \end{aligned}$$

# การหาวิถีสั้นสุดของทุกคู่ปม

## ❖ ใช้กำหนดการพลวัต

- ❖ ตรงไปตรงมา :  $\Theta(v^4)$
- ❖ repeated squaring :  $\Theta(v^3 \log v)$
- ❖ Floyd-Warshall :  $\Theta(v^3)$

## ❖ อัลกอริทึมของ Johnson

- ❖  $O(v \cdot e \log v)$  ดีสำหรับ sparse graph

# อัลกอริทึมของ Johnson

- ❖ ใช้ Dijkstra หา single-source shortest path ของทุกปม
  - ❖ linear array :  $O(v \cdot v^2) = O(v^3)$
  - ❖ binary heap :  $O(v \cdot e \log v)$
- ❖ แต่ Dijkstra ไม่รับ negative-weight edges
- ❖ ดังนั้น
  - ❖ เปลี่ยน G ให้เป็นกราฟใหม่ K ที่ไม่มีเส้นเชื่อมติดลบ (โดยวิถีสั้นสุดของ G ต้องเหมือนวิถีสั้นสุดของ K)
  - ❖ และใช้ Dijkstra จำนวน  $v$  ครั้งเพื่อหาวิถีสั้นสุดของทุกคู่ปม

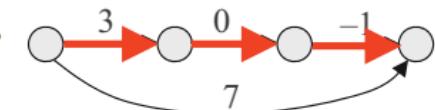
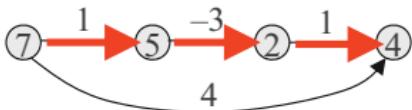
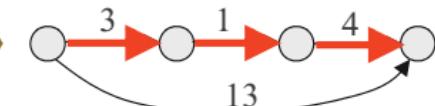
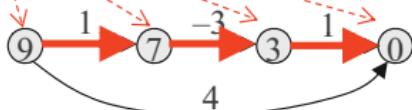
# การปรับความยาวเส้นเชื่อม



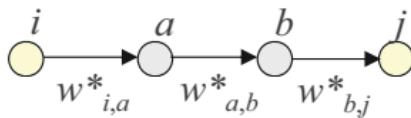
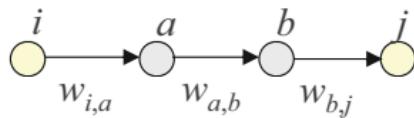
$$w_{i,j}^* = w_{i,j} + h(i) - h(j)$$

$h()$

$$1 + 9 - 7 = 3$$



# ความavar ใหม่ของวิธี



$$w^*_{i,a} = w_{i,a} + h(i) - h(a)$$

$$w^*_{a,b} = w_{a,b} + h(a) - h(b)$$

$$w^*_{b,j} = w_{b,j} + h(b) - h(j)$$

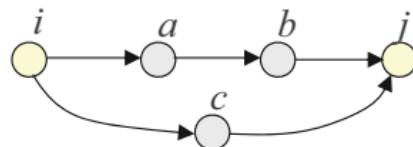
$$\begin{aligned} w^*_{i,a} + w^*_{a,b} + w^*_{b,j} &= w_{i,a} + w_{b,j} + w_{a,b} + \\ &\quad h(i) - h(a) + h(a) - h(b) + h(b) - h(j) \\ &= w_{i,a} + w_{b,j} + w_{a,b} + h(i) - h(j) \end{aligned}$$

$$w^*(i \rightarrow a \rightarrow b \rightarrow j) = w(i \rightarrow a \rightarrow b \rightarrow j) + h(i) - h(j)$$

# $h(i)$ มีค่าเท่าไรก็ได้ วิถีสั้นสุดไม่เปลี่ยน

## ❖ พิจารณาปม $i$ กับ $j$

❖ วิถีใด ๆ ของปมคู่นี้  
ถึงแม้ผ่านปมต่างกัน  
แต่มีปมปลายเหมือนกัน



❖ ความยาวใหม่ของวิถีใด ๆ ของปมคู่นี้ เพิ่มเป็นปริมาณเท่ากัน

$$w^*(i \rightarrow a \rightarrow b \rightarrow j) = w(i \rightarrow a \rightarrow b \rightarrow j) + h(i) - h(j)$$

$$w^*(i \rightarrow c \rightarrow j) = w(i \rightarrow c \rightarrow j) + h(i) - h(j)$$

## ❖ ดังนั้นวิถีสั้นสุด (หลังปรับความยาว) ไม่เปลี่ยน

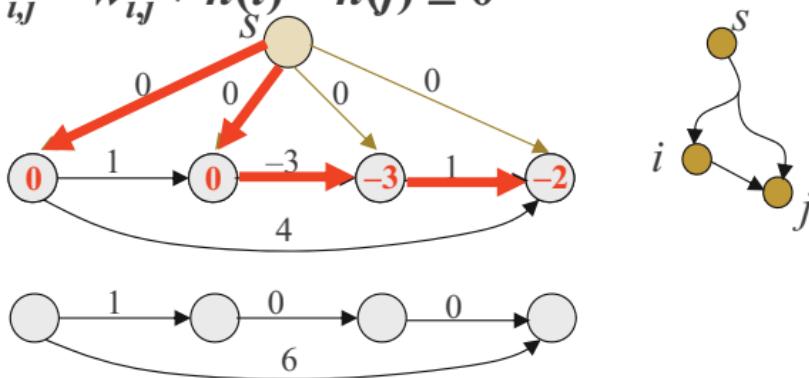
## ❖ ความยาวของวงในกราฟก็ไม่เปลี่ยน

$$w^*(d \rightarrow e \rightarrow f \rightarrow d) = w(d \rightarrow e \rightarrow f \rightarrow d) + h(d) - h(d)$$



# วิธีหา $h(i)$ ที่ทำให้ไม่มีเส้นเชื่อมติดลบ

- ❖ เดิมปม  $s$  และเส้นเชื่อมยาว 0 จาก  $s$  ถึงทุกปม
- ❖ หาวิถีสั้นสุดจาก  $s$  ถึงทุกปม (ใช้ Bellman-Ford)
- ❖ ให้  $h(i) =$  ความยาวของวิถีสั้นสุดจาก  $s$  ไป  $i$
- ❖ ถูเส้นเชื่อม  $(i, j)$  จะได้ว่า  $h(i) + w_{i,j} \geq h(j)$
- ❖ ดังนั้น  $w^*_{i,j} = w_{i,j} + h(i) - h(j) \geq 0$



# อัลกอริทึมของ Johnson

```
Johnson( G ) {  
    create K : K.V = G.V ∪ {s}  
        : K.E = G.E ∪ {(s,i) | i ∈ G.V)}  
        : K.w = G.w, K.w(s,i) = 0, i ∈ K.V)  
    h = BellmanFord(K, s) ← Θ(ve)  
    if (h == null) return "Negative Cycle"  
    else {  
        for each edge (i,j) ∈ G.E  
            G.w(i,j) += h[i] - h[j]  
        for each vertex i ∈ G.V {  
            d = Dijkstra(G, i) ← O(e log v)  
            for each vertex j ∈ G.V  
                D[i][j] = d[j] - (h[i] - h[j])  
        }  
        return D  
    }  
}
```

ถ้า Dijkstra ใช้ binary heap จะทำให้ Johnson ใช้เวลา  $O(v^e \log v)$  ซึ่งดีกว่า FloydWarshall เมื่อเทียบ sparse graph

## ลองคิดดู

- ❖ ถ้า  $G$  มีวง  $C$  ที่มีความยาวรวมของวงใน  $C$  เป็น  $0$ 
  - ❖ อยากรู้ว่า ความยาวใหม่ของเส้นเชื่อม  $w^*$  แต่ละเส้นใน  $C$  หลังการปรับความยาว จะเป็นเท่าใด
- ❖ จริงหรือไม่ ถ้า  $G$  เป็น **strongly connected graph**
  - ❖ เราไม่จำเป็นต้องเพิ่มปม  $s$  ตาม Johnson
  - ❖ แต่ใช้ปมใดก็ได้ใน  $G$  เป็นปมเริ่ม เพื่อใช้ Bellman-Ford หา  $h(i)$  ก็ได้



# การค้นปริภูมิสถานะ

สมชาย ประสิทธิ์จุตระกุล

มหาวิทยาลัยเทคโนโลยีราชมงคลรัตนโกสินทร์

# หัวข้อ

การแจกแจงผลเฉลยทุกรูปแบบ  
การค้นในแนวลึก  
การค้นในแนวกว้าง  
การย้อนรอย  
การค้นตามต้นทุนน้อยสุด  
การขยายและจำกัดเขต

# ลำดับการตัดสินใจ



## ❖ คำตอบได้มาจากการลำดับของการตัดสินใจ

- ❖ Sorting : สลับคู่ได้ก่อน ?
- ❖ Activity Selection : ทำกิจกรรมได้ก่อน ?
- ❖ MST : เลือกเส้นเชื่อมนำมาเป็นของ MST ?
- ❖ Knapsack : หยิบของขึ้นได้ใส่ถุงเป้ ?

## ❖ Greedy :

- ❖ วิธีเลือกแบบ greedy ที่ใช้นำไปสู่คำตอบที่ดีสุด (เจอกรณีแบบนี้ไม่น่า)

## ❖ Dynamic Programming

- ❖ หาคำตอบย่อยทุกแบบ เพื่อใช้ตัดสินใจสร้างคำตอบของปัญหาใหญ่
- ❖ เร็วเมื่อจำนวนปัญหาย่อยหักหมัดมีไม่น่า

## ❖ แต่มีปัญหามากมาย

- ❖ ไม่รู้วิธีเลือกแบบ greedy ที่ดีสุด
- ❖ จำนวนปัญหาย่อยมีมาก

จยหาทุกรูปแบบ  
แต่จะจยแบบฉลาด ๆ

# Sum of Subset (ที่เคยนำเสนอ)

- ❖ **input** :  $k$  และ  $D = \{d_1, d_2, d_3, \dots, d_n\}$   
 $k$  และ  $d_i$  เป็นจำนวนจริง
- ❖ **output** : true ถ้า  $D$  มีเซตย่อยที่ผลรวมเท่ากับ  $k$   
ถ้าไม่มี คืน false

$D = \{1, 4, 1, 9, 7\}, k = 11$  true

$D = \{1, 4, 1, 9, 7\}, k = 19$  false

# Sum of Subset (อีกแบบ)

- ❖ **input** :  $k$  และ  $D = \{d_1, d_2, d_3, \dots, d_n\}$   
 $k$  และ  $d_i$  เป็นจำนวนจริง
- ❖ **output** : หา  $S \subseteq D$  ที่ผลรวมของ  $S$  มีค่าเป็น  $k$

```
D = { 1, 4, 1, 9, 7 }, k = 11
```

```
S = { 1, 1, 9 }, {4, 7}
```

# รูปแบบของผลเฉลย

## ❖ Sum of subset

- ❖ output : หา  $S \subseteq D$  ที่ผลรวมของ  $S$  มีค่าเป็น  $k$

## ❖ คำตอบเป็นเซต

- ❖ ใช้ bit vector (อาร์เรย์ 1 มิติ)  $X = \langle x_1, x_2, x_3, \dots, x_n \rangle$
- ❖  $x_m = 1 \rightarrow d_m \in S$
- ❖  $x_m = 0 \rightarrow d_m \notin S$

## ❖ ตัวอย่าง

- ❖ input :  $D = \{2, 10, 3, 5, 7\}$ ,  $k = 14$
- ❖ output :  $X = \langle 1, 0, 0, 1, 1 \rangle$

## ❖ กำหนดรูปแบบของคำตอบให้แยกแจงได้ง่าย

# ลูกทุกเซตอยู่

- ❖ **input :  $D = \{2, 10, 3, 5, 7\}$ ,  $k = 14$**
- ❖ แจกแจงเซตอยู่ของ  $D$  ทั้งหมดมาตรวจสอบ
- ❖ คิดถึง **binary counter**
- ❖ ช้า : ต้องแจกแจงเซตอยู่  $2^n$  เซต

```

bcounter( n ) {
    x = new array[1..n] with all 0's
    for (i = 1; i <= 2n; i++) inc( x )
}
inc( x[1..n] ) {
    for (i=n; i>=1 AND x[i]==1; i--)
        x[i] = 0
    if (i >= 1) x[i] = 1
}

```

<0,0,0,0,0>  
<0,0,0,0,1>  
<0,0,0,1,0>  
<0,0,0,1,1>  
<0,0,1,0,0>  
<0,0,1,0,1>  
<0,0,1,1,0>  
<0,0,1,1,1>  
<0,1,0,0,0>  
.  
.  
.  
<1,1,1,0,1>  
<1,1,1,1,0>  
<1,1,1,1,1>

<0,0,1,1,1>  
↓  
↓  
↓  
↓  
↓  
↓  
<0,1,0,0,0>

# Sum of Subset

```
subsetSum( d[1..n] , k ) {  
    x = new array[1..n] with all 0's  
    for (i = 1; i <= 2n; i++) {  
        if (sum(d, x) == k) print( x )  
        inc( x )  
    }  
    inc( x[1..n] ) {  
        for (i=n; i>=1 AND x[i]==1; i--)  
            x[i] = 0  
        if (i >= 1) x[i] = 1  
    }  
}
```

$$\text{sum}(d, x) = \sum_{j=1}^n x[j]d[j]$$

# Binary Counter : Recursive

```

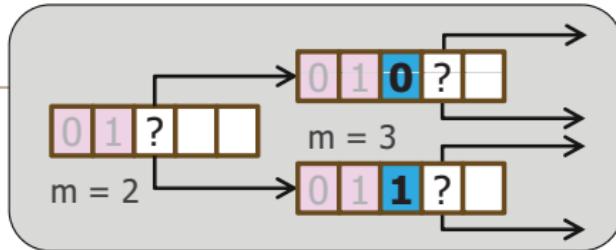
bcounter( n ) {
    x = new array[1..n] with all 0's
    count( x, 0 )
}

count( x[1..n], m ) {
    if ( m == n ) print( x )
    else {
        x[m+1] = 0; count( x, m+1 )
        x[m+1] = 1; count( x, m+1 )
    }
}

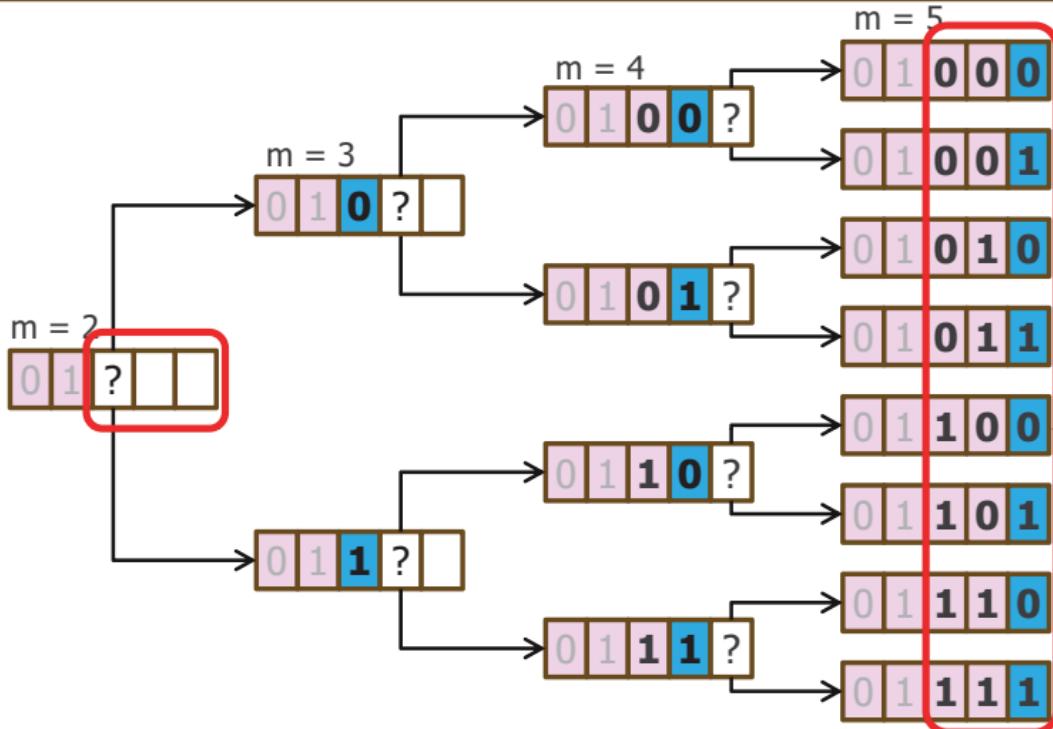
```

เติม  $x[1..m]$  และ  
จงเติมที่เหลือ

เติมครบแล้ว



# ขั้นตอนการเติม $x[1..n]$



# Sum of Subset : ลูยทุกเซตย่อๆ (#1)

```

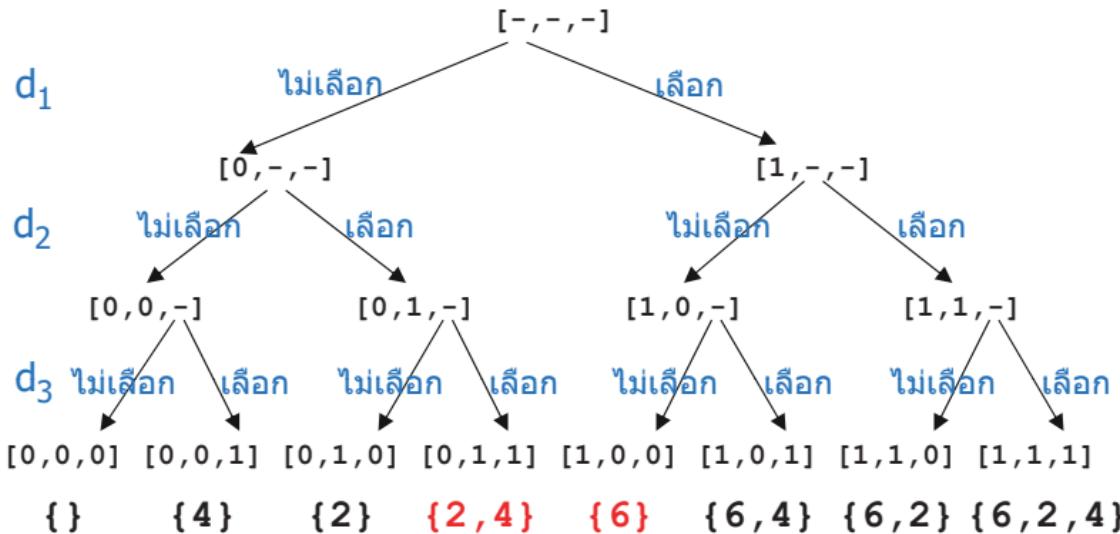
subsetSum( d[1..n] , k ) {
    x = new array[1..n] with all 0's
    subsetSum( d, k, x, 0 )
}

subsetSum( d[1..n] , k, x[1..n] , m ) {
    if (m == n) {
        if (sum(d, x) == k) print( x )
    } else {
        x[m+1] = 0; subsetSum( d, k, x, m+1 )
        x[m+1] = 1; subsetSum( d, k, x, m+1 )
    }
}

```

$$\Theta(n2^n)$$

$$D = \{6, 2, 4\}, k = 6$$



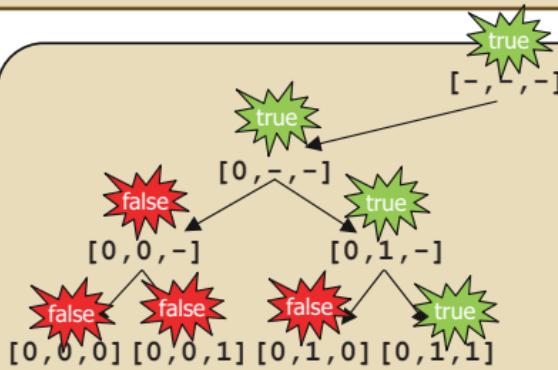
# ถ้าต้องการคำตอบเดียว

```

subsetSum( d[1..n] , k , x[1..n] , m ) {
    found = false
    if (m == n) {
        if (sum(d, x) == k) {
            print( x ); found = true
        }
    } else {
        x[m+1] = 0;
        found = subsetSum( d, k, x, m+1 )
        if (! found ) {
            x[m+1] = 1;
            - found = subsetSum( -d, k, x, m+1 )
        }
    }
    return found
}

```

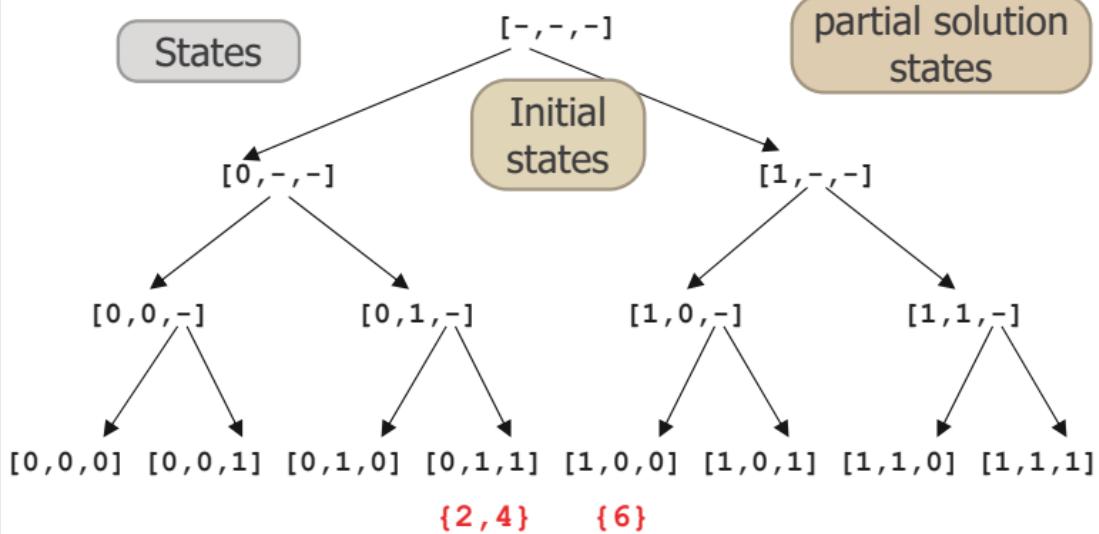
# ถ้าต้องการคำตอบเดียว



$$D = \{6, 2, 4\}, k = 6$$

$$\Sigma\{\} = 0 \quad \Sigma\{4\} = 4 \quad \Sigma\{2\} = 2 \quad \Sigma\{2, 4\} = 6$$

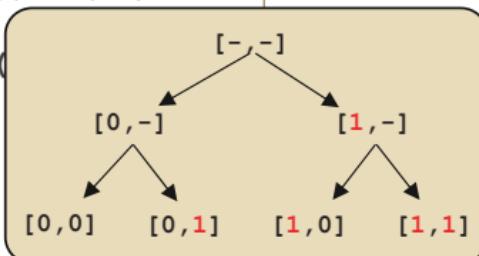
# ปริภูมิสถานะ (State Space)



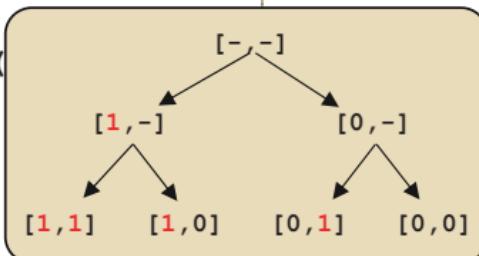
$D = \{6, 2, 4\}, k = 6$   
ค่าตอบคือ  $\{2, 4\}, \{6\}$

# ปริภูมิสถานะมีได้หลายแบบ

```
subsetSum( d[1..n], k, x[1..n], m ) {  
    if (m == n) {  
        if (sum(d, x) == k) print()  
    } else {  
        x[m+1] = 0; subsetSum( d,  
        x[m+1] = 1; subsetSum( d,
```



```
subsetSum( d[1..n], k, x[1..n], m ) {  
    if (m == n) {  
        if (sum(d, x) == k) print()  
    } else {  
        x[m+1] = 1; subsetSum( d,  
        x[m+1] = 0; subsetSum( d,
```

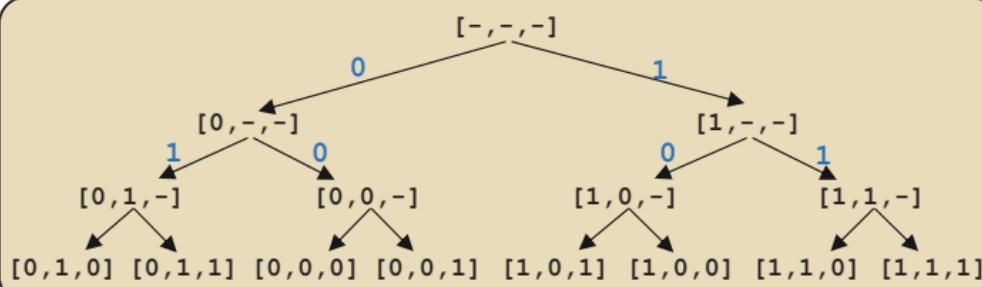


# ปริภูมิสถานะมีได้หลายแบบ

```

subsetSum( d[1..n], k, x[1..n], m ) {
    if (m == n) {
        if (sum(d, x) == k) print( x )
    } else {
        x[m+1] = random( {0,1} ) // 0 or 1
        subsetSum( d, k, x, m+1 )
        x[m+1] = 1 - x[m+1]
        subsetSum( d, k, x, m+1 )
    }
}

```



# ปริภูมิสถานะมีได้หลายแบบ

```
subsetSum( d[1..n], k, x[1..n], m ) {  
    if (m == n) {  
        if (sum(d, x) == k) print( x )  
    } else {  
        x[m+1] = 0; subsetSum( d, k, x, m+1 )  
        x[m+1] = 1; subsetSum( d, k, x, m+1 )  
    }  
}
```

m = 2 

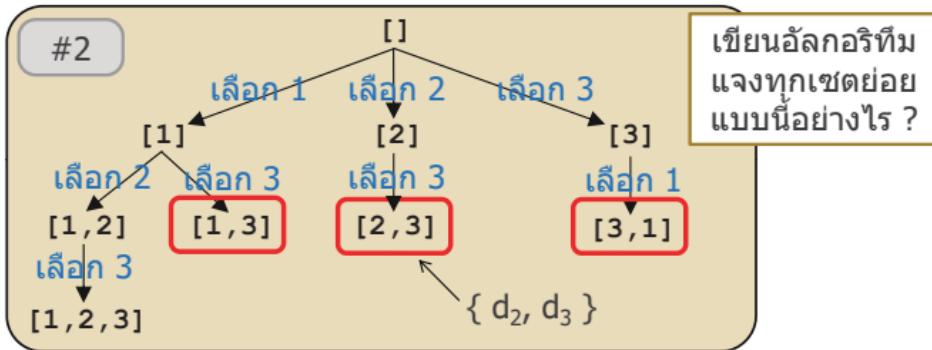
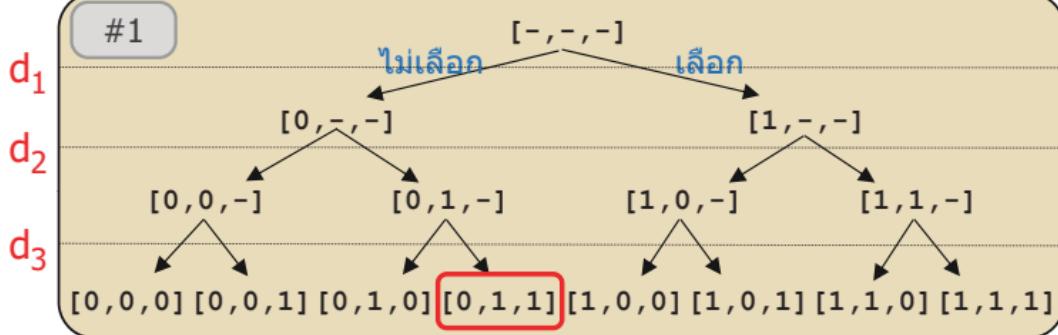


```
subsetSum( d[1..n], k, x[1..n], m ) {  
    if (m == n) {  
        if (sum(d, x) == k) print( x )  
    } else {  
        x[n-m] = 0; subsetSum( d, k, x, m+1 )  
        x[n-m] = 1; subsetSum( d, k, x, m+1 )  
    }  
}
```

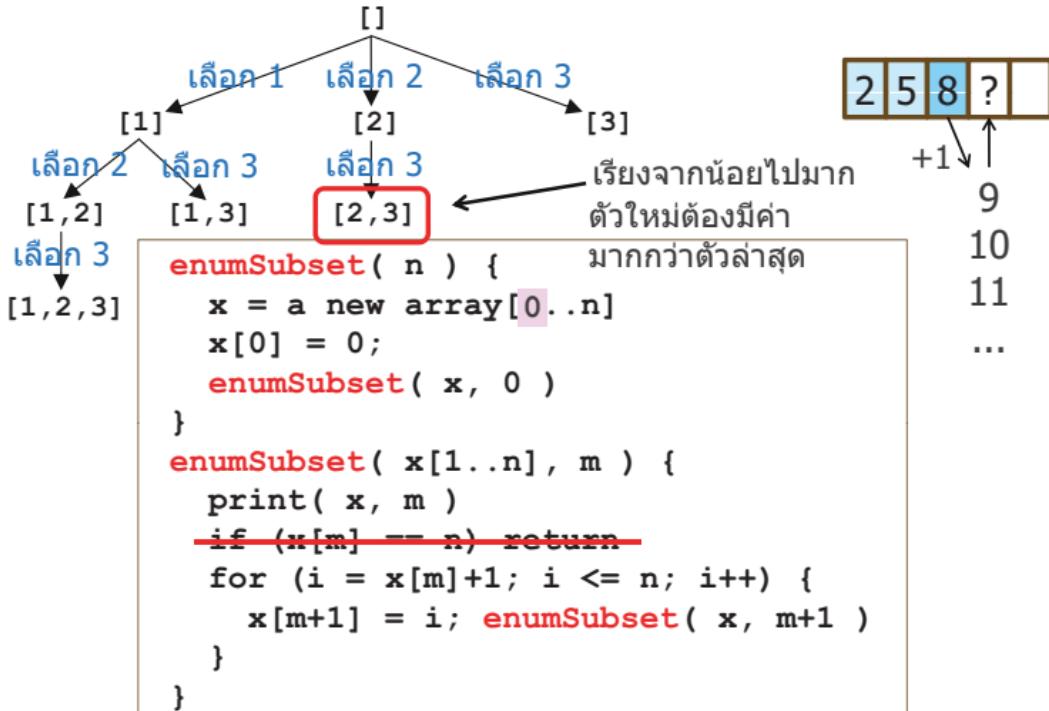
m = 2 



# ปริภูมิสถานะมีได้หลายแบบ



## การแจกทุกเซตย่อย (#2)



# Sum of Subset : ลูยทุกเซตป้อง (#2)

```

subsetSum( d[1..n] , k ) {
    x = a new array[0..n]
    x[0] = 0
    subsetSum( d, k, x, 0 )
}

subsetSum( d[1..n], k, x[0..n], m ) {
    if (sum(d, x, m) == k) print( x )
    for (i = x[m]+1; i <= n; i++) {
        x[m+1] = i;
        subsetSum( d, k, x, m+1 )
    }
}

```

$$\text{sum}(d, x, m) = \sum_{j=1}^m d[x[j]]$$

$x = [1, 2, 5, -\dots]$   
 $d[1] + d[2] + d[5]$

# 0/1 Knapsack

- ❖ ของ  $n$  ชิ้นมีหมายเลข :  $1, 2, 3, \dots, n$
- ❖ แต่ละชิ้นหนัก :  $w_1, w_2, w_3, \dots, w_n$
- ❖ แต่ละชิ้นมีมูลค่า :  $v_1, v_2, v_3, \dots, v_n$
- ❖ ถุงเป้นนึงใบจุของได้นักไม่เกิน  $W$
- ❖ ปัญหา : จงเลือกของใส่ถุง เพื่อให้
  - ❖ ถุงไม่ขาด
  - ❖ ได้มูลค่ารวมมากสุด

1	2.5Kg. \$400
2	1.4Kg. \$50
3	2.8Kg. \$350
4	0.7Kg. \$150
	4Kg.

เขียน state space ของ  
ปัญหานี้ (มีของ 4 ชิ้น)

เขียน pseudo code ที่หา  
คำตอบของปัญหานี้

# K - Coloring

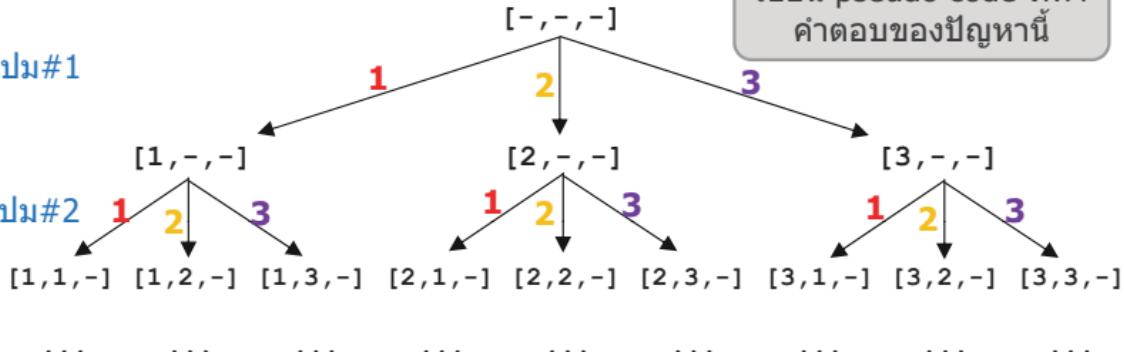
- ❖ **Input** : กราฟ  $G = (V, E)$ , จำนวนเต็ม  $k$
- ❖ **Output** : หาวิธีให้สีกับปมต่าง ๆ ใน  $G$  ที่
  - ❖ ใช้ไม่เกิน  $k$  สี
  - ❖ ปมปลายของเส้นเชื่อมเดียวกัน  
ห้ามมีสีเดียวกัน

เขียน state space ของปัญหานี้  
(กราฟมี 4 ปม,  $k=3$ )

เขียน pseudo code ที่หา  
คำตอบของปัญหานี้

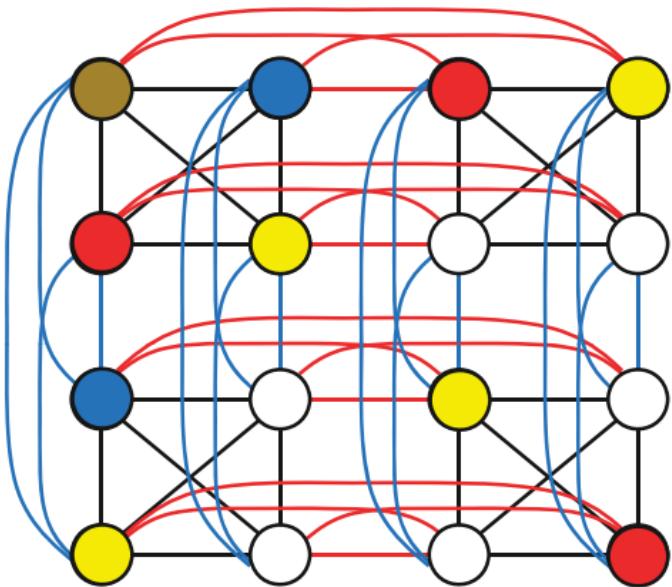
ปม #1

ปม #2



# Sudoku → K-Coloring

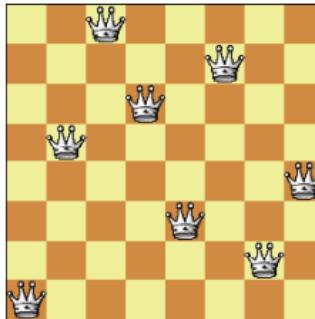
1	2	3	4
3	4		
2			4
4			3



## n-Queen

❖ ปัญหา : วางควีน n ตัวบนตาราง  $n \times n$

❖ ไม่มีตัวใดเห็นตัวใดเลยในแนวตั้ง แนวนอน และแนวทแยง



เขียน state space ของ  
4-queen



**Depth - First Search**

**Breadth - First Search**

# DFS กับ BFS

- ❖ คล้ายกับ DFS และ BFS ของการค้นปมในกราฟ
- ❖ แต่ในที่นี้ เราค้นในปริภูมิสถานะ
  - ❖ ไม่มีโครงสร้างมาให้ค้น
  - ❖ ปมสถานะถูกสร้างขึ้นระหว่างการค้น
  - ❖ เส้นเชื่อมระหว่างปม คือการผลิตปมใหม่จากปมเดิม
- ❖ ถ้าเขียนแบบเรียกช้า
  - ❖ stack frame ที่เก็บพารามิเตอร์ต่าง ๆ ของการเรียกช้า แทนสถานะของการค้น
- ❖ ถ้าเขียนแบบวงวนทำช้า
  - ❖ ต้องสร้างที่เก็บข้อมูล ไว้แทนสถานะของการค้น

# Depth-First Search : Recursive

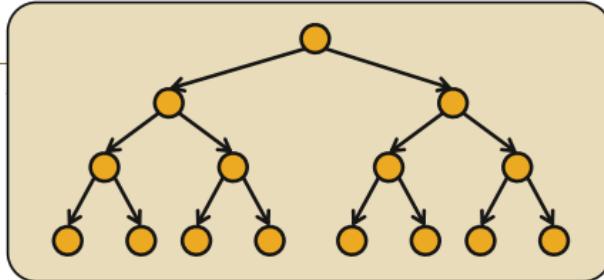
```

subsetSum( d[1..n], k ) {
    x = new array[1..n] with all 0's
    subsetSum( d, k, x, 0 )
}

subsetSum( d[1..n], k, x[1..n], m ) {
    if (m == n) {
        if (sum(d, x) == k) print( x )
    } else {
        x[m+1] = 0; subsetSum( d, k, x, m+1 )
        x[m+1] = 1; subsetSum( d, k, x, m+1 )
    }
}

```

พารามิเตอร์คือ  
สถานะของการค้น



# Depth-First Search : Iterative

```
subsetSum( d[1..n] , k )
S ← an empty stack;
S.push( a zero-length array )
while ( S ≠ ∅ ) {
    x[1..m] ← S.pop()
    if ( m == n ) {
        if (sum(d, x) == k) {
            print(x); break
        }
    } else {
        x0 = copyOf(x, m+1); x0[m+1] = 0
        x1 = copyOf(x, m+1); x1[m+1] = 1
        S.push(x1);
        S.push(x0);
    }
}
}

```

แทนสถานะด้วย array ที่เก็บใน stack

สร้างอาร์เรย์ขนาด  $m+1$  มี  
ข้อมูลเหมือนกับของ  $x$

# Breadth-First Search

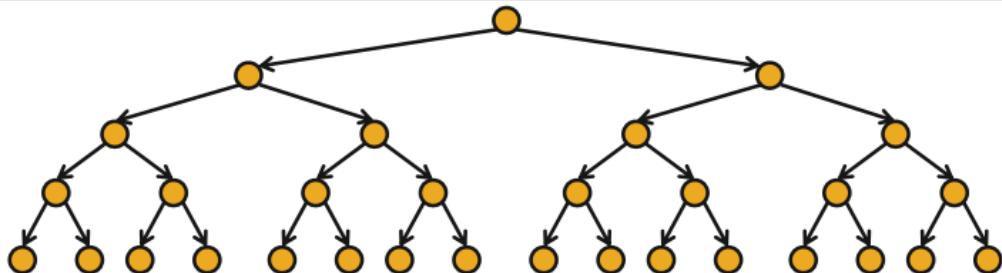
```

subsetSum( d[1..n], k )
Q ← an empty queue;
Q.enqueue( a zero-length array )
while ( Q ≠ ∅ ) {
    x[1..m] ← Q.dequeue()
    if ( m == n ) {
        if (sum(d, x) == k) {
            print(x); break
        }
    } else {
        x0 = copyOf(x, m+1); x0[m+1] = 0
        x1 = copyOf(x, m+1); x1[m+1] = 1
        Q.enqueue(x0);
        Q.enqueue(x1);
    }
}

```

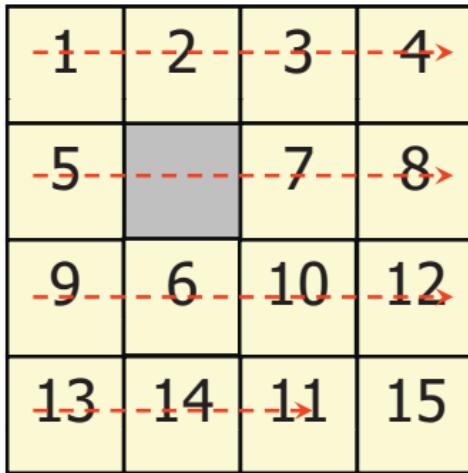
แทนสถานะด้วย array ที่เก็บใน queue

# ปริมาณหน่วยความจำ DFS - BFS



- ❖ 1 ปมแต่ก  $b$  กิ่ง ต้นไม้สูง  $h$
- ❖ **depth-first search**
  - ❖ ใช้ stack เนื้อที่  $O(bh)$
- ❖ **breadth-first search**
  - ❖ ใช้ queue เนื้อที่  $O(b^h)$

# 15-puzzle

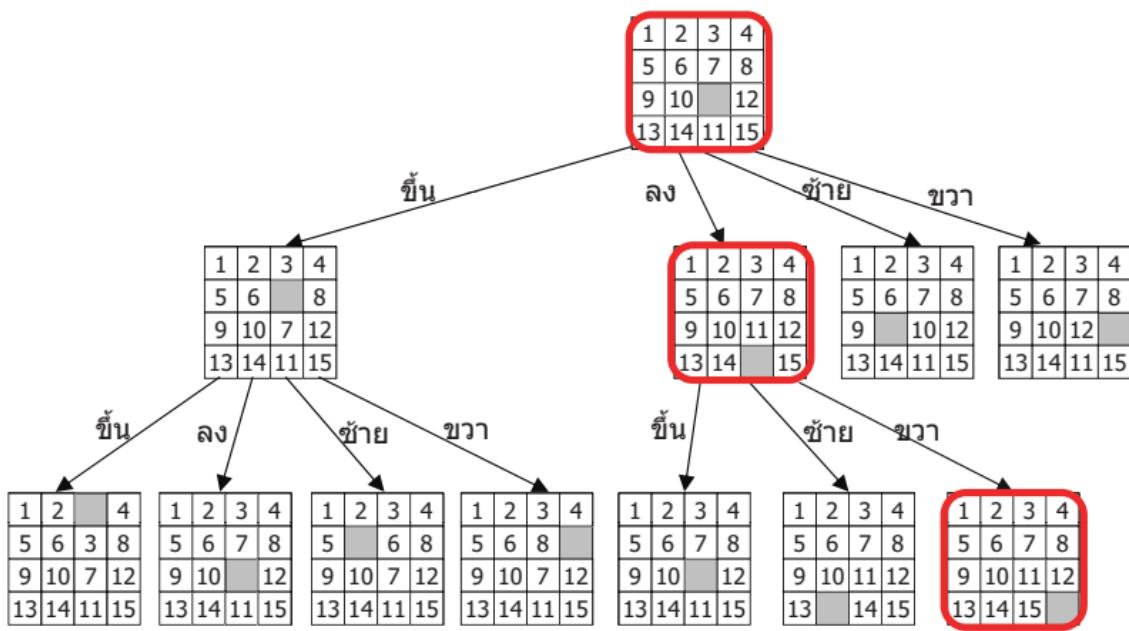


# 15-puzzle

1	2	3	4
5	6	7	8
9	10	12	
13	14	11	15



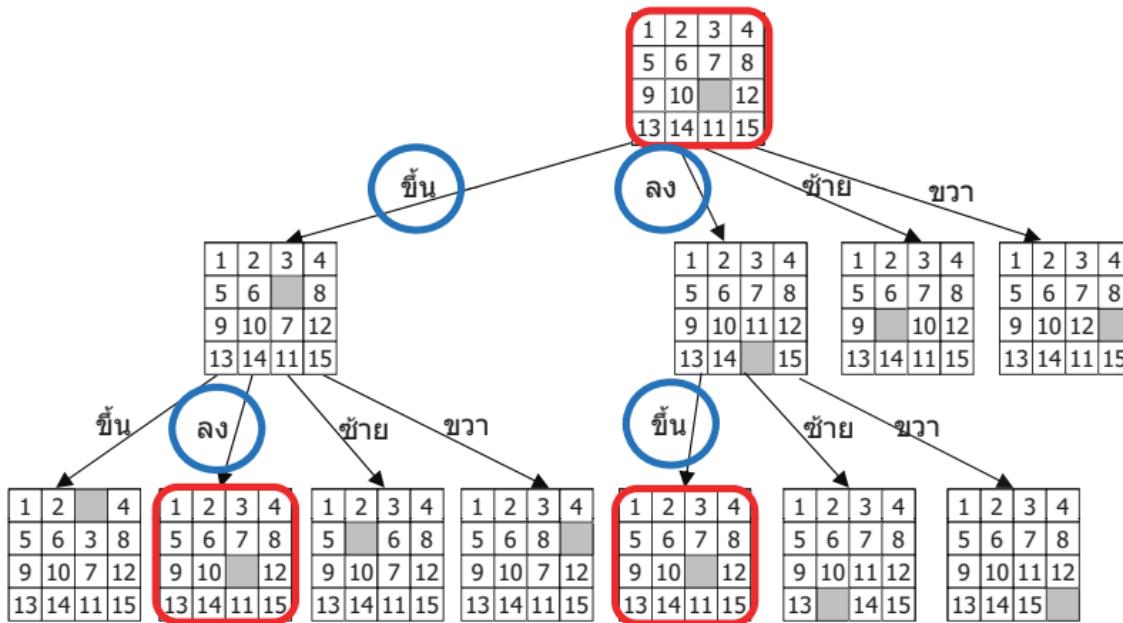
# ค้นตามแนวกว้าง



# 15-puzzle : Breadth-First Search

```
solve( board ) {  
    Q = an empty queue  
    Q.enqueue( board );  
    while ( Q ≠ Ø ) {  
        b = Q.dequeue();  
        if ( b.isAnswer() ) return b;  
        for each d ∈ {UP, DOWN, LEFT, RIGHT} {  
            if ( b.canMoveBlank( d ) ) {  
                b1 = b.moveBlank( d );  
                Q.enqueue( b1 );  
            }  
        }  
    }  
    print "no solution";  
}
```

## ระวังการผลิตปมสถานะที่ช้าช่อน



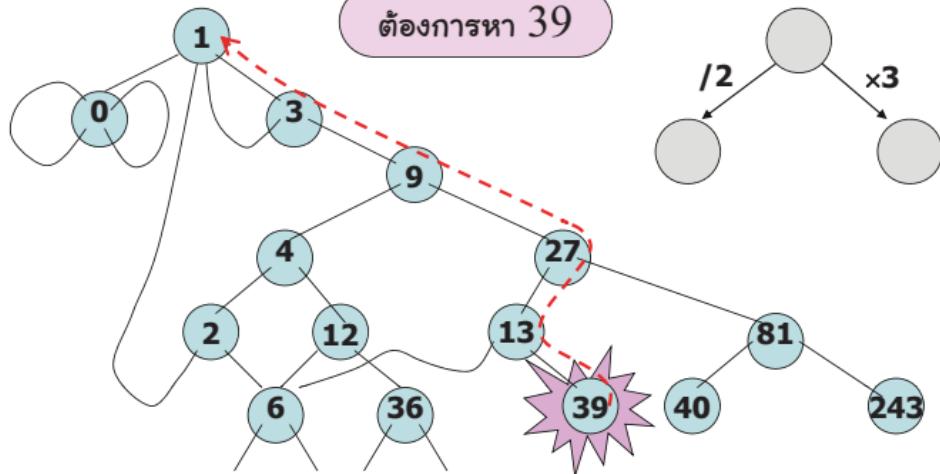
# ใช้ set เก็บปมสถานะต่าง ๆ ที่เคยผลิต

```
solve(board) {  
    set = an empty set  
    Q = an empty queue  
    Q.enqueue( board ); set.add( board )  
    while ( Q ≠ Ø ) {  
        b = Q.dequeue();  
        if ( b.isAnswer() ) return b;  
        for each d ∈ {UP, DOWN, LEFT, RIGHT} {  
            if ( b.canMoveBlank( d ) ) {  
                b1 = b.moveBlank( d );  
                if (b1 ∉ set) {  
                    Q.enqueue( b1 ); set.add( b1 )  
                }  
            }  
        }  
    }  
    print "no solution";  
}
```

# ตัวอย่าง : คูณสามหารสอง

- ❖ ให้จำนวนเต็ม  $v$
- ❖ เริ่มด้วย 1 จะต้อง  $\times 3$  และหรือ  $/2$  (ปิดเศษทิ้ง)  
อย่างไร จึงมีค่าเท่ากับ  $v$
- ❖ เช่น
  - ❖  $v = 10 = 1 \times 3 \times 3 \times 3 \times 3 / 2 / 2 / 2$
  - ❖  $v = 31 = 1 \times 3 \times 3 \times 3 \times 3 \times 3 / 2 / 2 / 2 / 2 \times 3 \times 3 / 2$
- ❖ ขอจุยทุกรูปแบบตามแนวว้าง

# คูณสามหารสอง : ค้นตามแนวกว้าง



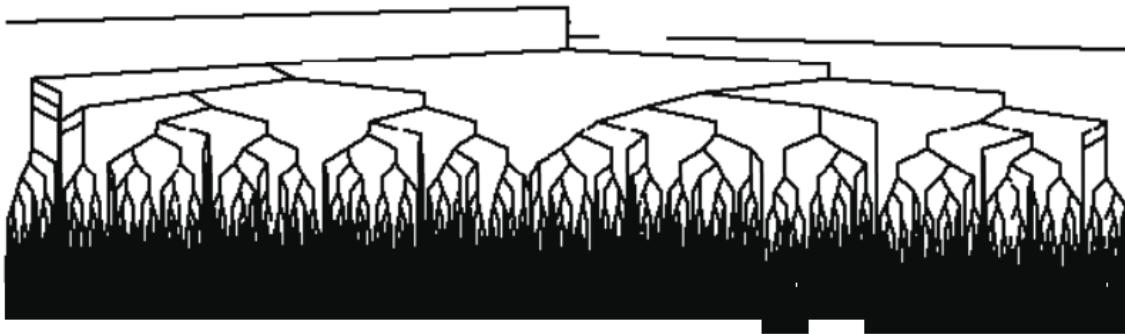
$$39 = 1 \times 3 \times 3 \times 3 / 2 \times 3$$

→  
BFS : สั้นสุด

# คุณสามหารสอง : ค้นตามแนวกว้าง

```
m3d2( target ) {  
    set = an empty set  
    Q = an empty queue  
    state = new State(1);  
    Q.enqueue( state ); set.add( state )  
    while ( Q ≠ Ø ) {  
        state = Q.dequeue();  
        if (state.value == target) return state;  
        d2 = new State(state.value/2, state);  
        if (d2 ∉ set) {Q.enqueue( d2 ); set.add( d2 )}  
        m3 = new State(state.value*3, state);  
        if (m3 ∉ set) {Q.enqueue( m3 ); set.add( m3 )}  
    }  
    return Ø  
}
```

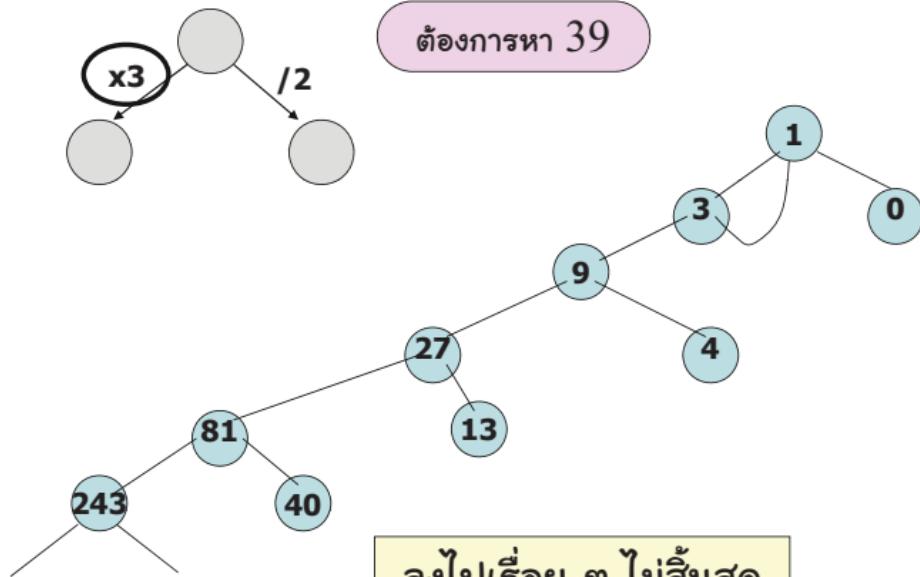
# BFS : 41 = ?



การค้นในแนวกว้างได้ต้นไม้ 12628 ปม สูง 23

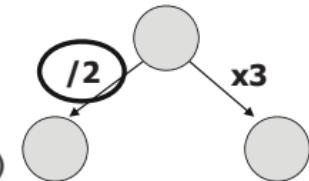
$$\begin{aligned} 41 = & \ 1 \times 3 \times 3 \times 3 / 2 \times 3 \times 3 / 2 / 2 \times 3 \times 3 \\ & / 2 / 2 \times 3 \times 3 / 2 / 2 / 2 \times 3 \times 3 / 2 / 2 / 2 / 2 \end{aligned}$$

# ค้นตามแนวลึก อาจหาคำตอบไม่พบ

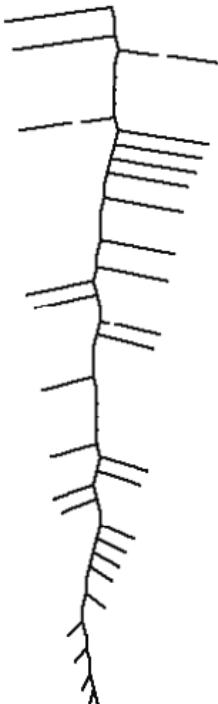


# คูณสามหารสอง : ค้นตามแนวลึก

```
m3d2( target ) {  
    set = an empty set  
    S = an empty stack  
    state = new State(1);  
    S.push( state ); set.add( state )  
    while ( S ≠ Ø ) {  
        node = S.pop();  
        if (state.value == target) return state;  
        m3 = new State(state.value*3, state);  
        if (m3 ∉ set) {S.push( m3 ); set.add( m3 )}  
        d2 = new State(state.value/2, state);  
        if (d2 ∉ set) {S.push( d2 ); set.add( d2 )}  
    }  
    return Ø  
}
```



# DFS : 41 = ?



**DFS** : ได้ต้นไม้ 82 ปม สูง 51

$$41 = 1 \times 3 \times 3 / 2 / 2 \times 3 \times 3 \times 3 / 2 / 2 \\ / 2 / 2 \times 3 / 2 \times 3 \times 3 / 2 \times 3 / 2 \\ / 2 / 2 \times 3 / 2 \times 3 \times 3 / 2 \times 3 \times 3 \\ \times 3 \times 3 / 2 / 2 / 2 / 2 \times 3 / 2 / 2 \\ / 2 / 2 \times 3 / 2 \times 3 / 2 \times 3 / 2 \times 3 / 2$$

**BFS** : ได้ต้นไม้ 12628 ปม สูง 23

$$41 = 1 \times 3 \times 3 \times 3 / 2 \times 3 \times 3 / 2 / 2 \times 3 \times 3 \\ / 2 / 2 \times 3 \times 3 / 2 / 2 / 2 \times 3 \times 3 / 2 \\ / 2 / 2 / 2$$

# คูณสามหารสอง : ค้นตามแนวลึก

```
m3d2( target ) {
    return "1" + m3d2(target, 1, new empty set)
}
m3d2( target, v, set ) {
    if (v == target) return ""
    d2 = v / 2;
    if (d2 ∉ set) {
        set.add(d2); result = m3d2(target, d2, set)
        if (result != Ø) return "/2" + result
    }
    m3 = v * 3;
    if (m3 ∉ set) {
        set.add(m3); result = m3d2(target, m3, set);
        if (result != Ø) return "x3" + result
    }
    return Ø
}
```

# DFS vs. BFS

## ❖ depth-first search ใช้ stack

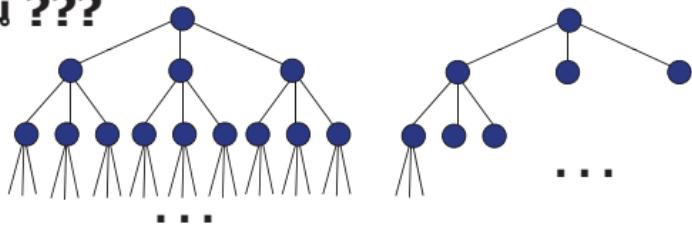
- ❖ ถ้าต้นไม้มีขนาดไม่จำกัด, อาจหาไม่พบค่าตอบ
- ❖ ต้นไม้สูง  $h$ , จะใช้เนื้อที่ใน stack แค่  $O(b \cdot h)$

## ❖ breadth-first search ใช้ queue

- ❖ ประกันว่าพบ answer state ที่ใกล้รากสุด
- ❖ ต้นไม้สูง  $h$ , จะใช้เนื้อที่ใน queue เป็น  $O(b^h)$

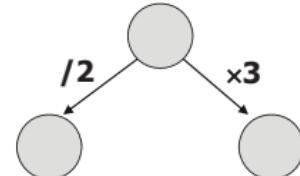
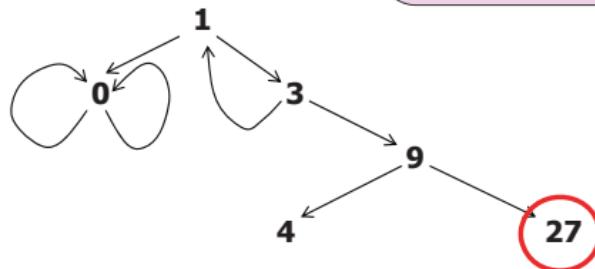
## ❖ state space tree มีขนาดใหญ่, ซ้ำมาก ๆ

## ❖ มีข้อดีตรงไหน ???



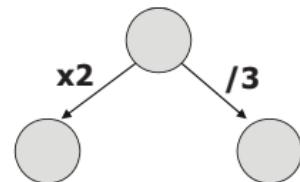
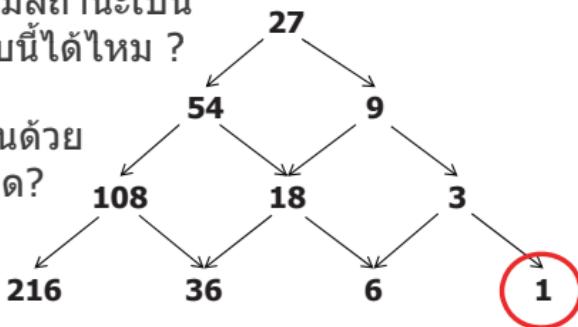
# คูณสามหารสอง

ต้องการหา 27



ปริภูมิสถานะเป็น  
แบบนี้ได้ไหม ?

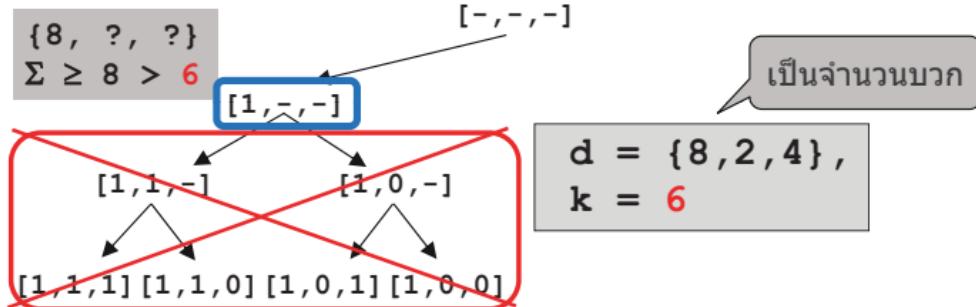
ต้องค้นด้วย  
วิธีใด?



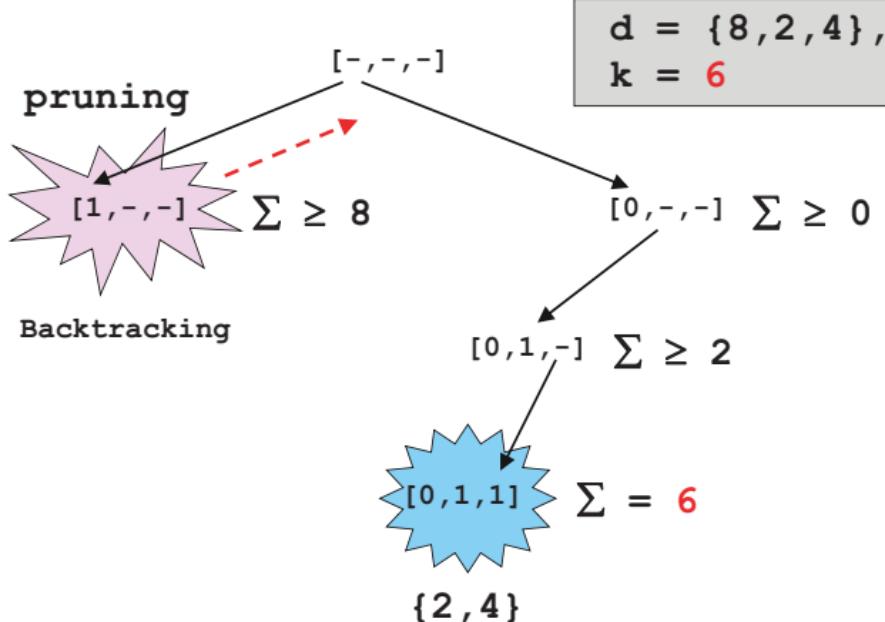
# Backtracking

# การย้อนรอย (backtracking)

- ❖ ค้นทุกปมในปริภูมิสถานะ → ข้า
- ❖ พิจารณาสถานะที่ແນ່ນຜລເລຍນາງສົວນ (partial solution state) ຄໍາໄມ່ມີແວວ ໄມ່ຄັນຕ່ອ
- ❖ ໄມ່ມີແວວ → ມັນໃຈວ່າໄມ່ພບສຖານະຄຳດອບແນ່ ໆ



# Sum of Subset : Backtracking



# DFS + Backtracking

```
subsetSum( d[1..n], k, x[1..n], m ) {  
    sum = sum(d, x, m)  
    if (sum > k) return  
    if (m == n) {  
        if (    sum == k      ) print( x )  
    } else {  
        x[m+1] = 1; subsetSum( d, k, x, m+1 )  
        x[m+1] = 0; subsetSum( d, k, x, m+1 )  
    }  
}
```

$$\text{sum}(d, x, m) = \sum_{j=1}^m x[j]d[j]$$

# DFS + Backtracking

```

subsetSum( d[1..n], k, x[1..n], m ) {
    sum = sum(d, x, m)
    if (sum > k) return
    if (m == n) {
        if ( sum == k ) print( x )
    } else {
        if ( sum+d[m+1] <= k )
            x[m+1] = 1; subsetSum( d, k, x, m+1 )
        x[m+1] = 0; subsetSum( d, k, x, m+1 )
    }
}

```

$$\text{sum}(d, x, m) = \sum_{j=1}^m x[j]d[j]$$

# BFS + Backtracking

```
subsetSum( d[1..n], k )
Q ← an empty queue;
Q.enqueue( a zero-length array )
while ( Q ≠ ∅ ) {
    x[1..m] ← Q.dequeue()
    sum = sum(d, x, m)
    if (sum ≤ k) {
        if ( m == n ) {
            if ( sum == k ) print(x)
        } else {
            x0 = copyOf(x, m+1); x0[m+1] = 0
            x1 = copyOf(x, m+1); x1[m+1] = 1
            Q.enqueue(x0);
            Q.enqueue(x1);
        }
    }
}
```

# Sum of Subset : Backtracking

n	จำนวนคำตอบ	DFS (จำนวนปมนี่คัน)		DFS + Backtracking (จำนวนปมนี่คัน)	
		คำตอบแรก	คำตอบทั้งหมด	คำตอบแรก	คำตอบทั้งหมด
10	6	1,402	2,047	133	384
11	20	621	4,095	97	1,934
12	37	708	8,191	78	3,660
13	36	1,922	16,383	144	4,100
14	81	3,967	32,767	26	8,198
15	204	1,797	65,535	60	27,351
16	128	4,085	131,071	41	20,015
17	616	7,684	262,143	76	70,795
18	1,052	16,354	524,287	74	121,853
19	2,515	6,649	1,048,575	120	360,480
20	7,133	6,152	2,097,151	21	949,847
21	6,837	30,472	4,194,303	97	789,842
22	8,979	442,309	8,388,607	139	915,099
23	50,129	14,090	16,777,215	73	7,145,916
24	92,457	24,456	33,554,431	25	12,847,798
25	54,493	1,015,288	67,108,863	234	6,159,965

# ลองคิดดู : Sum of Subset

- ต้องปรับอย่างไร ถ้าเซตเก็บจำนวนลบได้

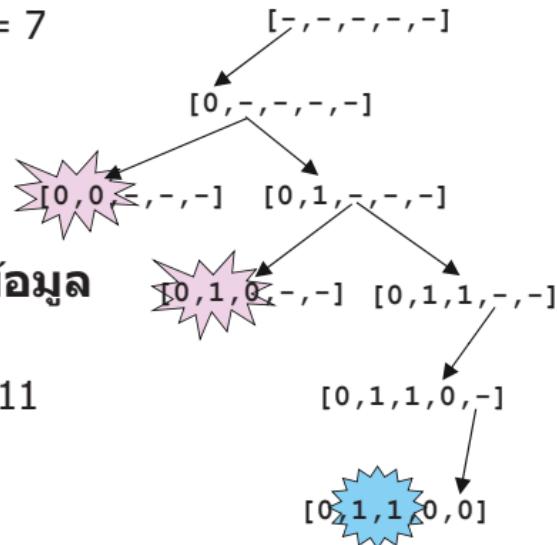
❖  $d = \{-3, 4, -2, 5, 1\}$ ,  $k = 7$

- ของเดิม : เราไม่ลงต่อ

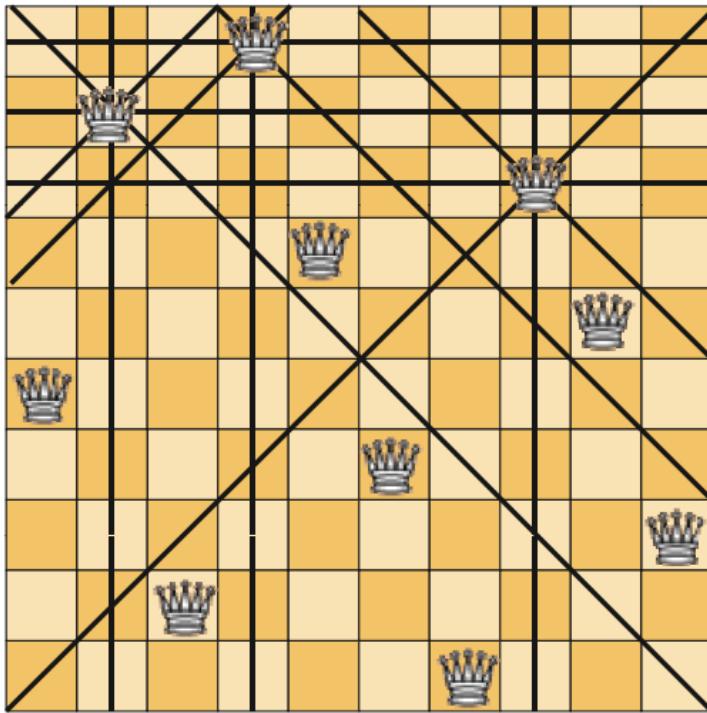
$$\text{ถ้า } \sum_{j=1}^m x[j]d[j] > k$$

ของใหม่ : เรียงลำดับข้อมูล  
จะปรับอย่างไรให้ดีขึ้น

❖  $d = \{8, 6, 5, 2, 1\}$ ,  $k = 11$



# n-Queen Problem



# ใช้อาร์ย์ 1 มิติแทนการวางควีน

แถวที่ 1 วางควอลัม์ที่ 2

col	1	2	3	4	...
1	2				
2		Q			
3				Q	
4				Q	
5	1				
.					
.					

col [r] แทนหมายเลขคอลัมน์ที่วางควีนของแถวที่ r

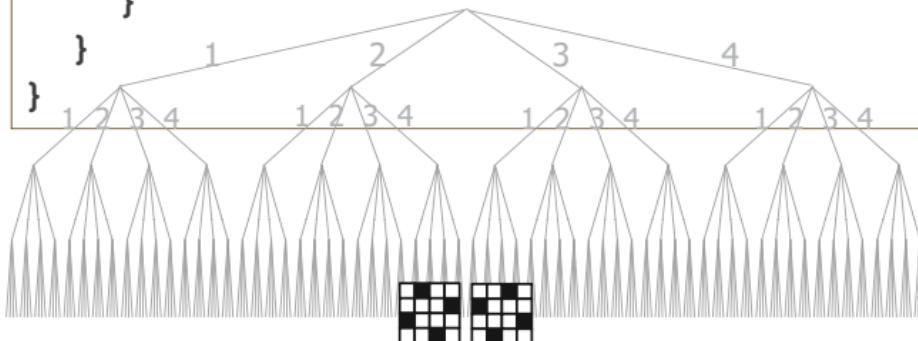
[2, 4, 9, 3, 1, ... ]

# n-Queen : DFS

```

queen(col[1..n], m) {
    if (m == n) {
        if (isValid(col)) print(col)
    } else {
        for (i = 1; i <= n; i++) {
            col[m+1] = i; queen(col, m+1);
        }
    }
}

```



[2,4,1,3]      [3,1,4,2]

[1, 1, 1, 1]  
 [1, 1, 1, 2]  
 [1, 1, 1, 3]  
 [1, 1, 1, 4]  
 [1, 1, 2, 1]  
 [1, 1, 2, 2]  
 [1, 1, 2, 3]  
 [1, 1, 2, 4]

. . .

[4, 4, 3, 1]  
 [4, 4, 3, 2]  
 [4, 4, 3, 3]  
 [4, 4, 3, 4]  
 [4, 4, 4, 1]  
 [4, 4, 4, 2]  
 [4, 4, 4, 3]  
 [4, 4, 4, 4]

# การวางแผนที่ผิดกฎหมาย

[1,4,1,3]

	1		
4			4
1	1		
3		3	

คอลัมน์เดียวกัน

มีตัวเลขซ้ำกัน  
หรือไม่ ?

[1,4,2,3]

1	1		
4			4
2	2		
3		3	

แนวทแยงเดียวกัน

[2,?,-,3]

2	2		
?	1		4
-			
3		3	

ไม่เกิด

[1, 1, 1, 1]

[1, 1, 1, 2]

[1, 1, 1, 3]

...

[4, 4, 4, 1]

[4, 4, 4, 2]

[4, 4, 4, 3]

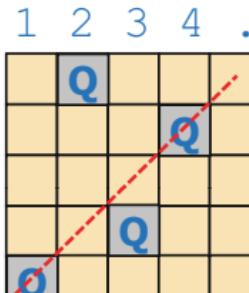
[4, 4, 4, 4]

[2,4,1,3]

2	2		
4			4
1	1		
3		3	

# การตรวจสอบแนวทแยง

col  
 1    2    3    4    ...  
 2    4    9    3     
 4    3    1    5     
 5    1    2    4     
 :    col[r]



แทนหมายเลขคอลัมน์  
ที่วางควีนของقاءที่  $r$

$$\begin{aligned}
 |1 - 2| &\neq |2 - 4| \\
 |1 - 3| &\neq |2 - 9| \\
 |1 - 4| &\neq |2 - 3| \\
 |1 - 5| &\neq |2 - 1| \\
 &... \\
 |2 - 3| &\neq |4 - 9| \\
 |2 - 4| &\neq |4 - 3| \\
 |2 - 5| &= |4 - 1|
 \end{aligned}$$

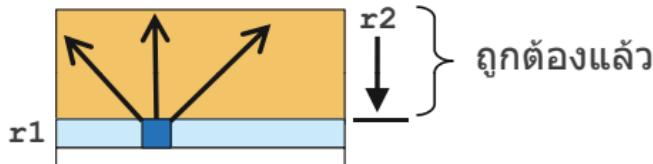


ควีนของقاء  $r_1$  กับ  $r_2$  จะไม่เห็นกันแนวทแยง ถ้า

$$|r_1 - r_2| \neq |\text{col}[r_1] - \text{col}[r_2]|$$

# การตรวจสอบ n-Queen ที่ถูกต้อง

```
isValid(col[1..n]) {  
    for (r1 = 2; r1 <= n; r1++) {  
        for (r2 = 1; r2 < r1; r2++) {  
            if ( col[r1] == col[r2] ||  
                |col[r1] - col[r2]| == r1 - r2 )  
                return false  
        }  
        return true;  
    }  
}
```



คุณของแقاء  $r_1$  กับ  $r_2$  จะไม่เห็นกันแนวแท้ถ้า

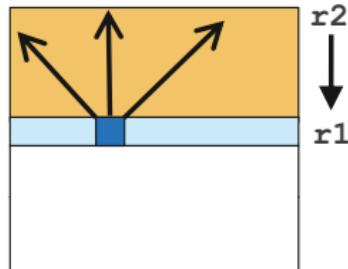
$$|r_1 - r_2| \neq |col[r_1] - col[r_2]|$$

# n-Queen : DFS + Backtracking

```

queen(col[1..n], m) {
    if (m == n) {
        if (isValid(col)) print(col)
    } else {
        for (i = 1; i <= n; i++) {
            col[m+1] = i
            if (isValid(col, m+1))
                queen(col, m+1)
        }
    }
}

```

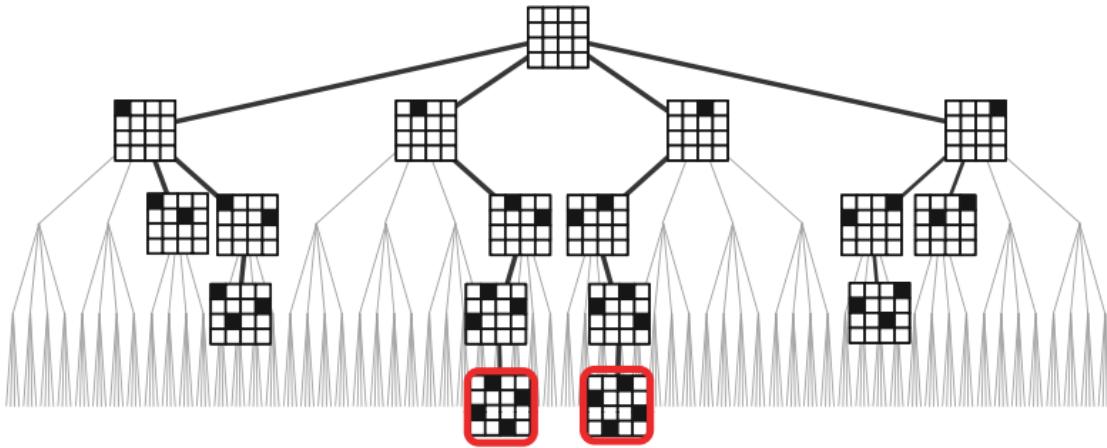


```

isValid(col[1..n], k) {
    r1 = k
    for (r2 = 1; r2 < r1; r2++) {
        if (col[r1] == col[r2] ||
            |col[r1] - col[r2]| == r1 - r2 )
            return false
    }
    return true;
}

```

# **n-Queen : DFS + Backtracking**



**n-Queen**

<b>n</b>	จำนวน คำตอบ	จำนวนการ เรียกข้ามพน คำตอบแรก
4	2	9
5	10	6
6	4	32
7	40	10
8	92	114
9	352	42
10	724	103
11	2,680	53
12	14,200	262
13	73,712	112
14	365,596	1,900
15	2,279,184	1,360
16	14,772,512	10,053
17	95,815,104	5,375

# 0/1 Knapsack

- ❖ ของ  $n$  ชิ้นมีหมายเลข :  $1, 2, 3, \dots, n$
- ❖ แต่ละชิ้นหนัก :  $w_1, w_2, w_3, \dots, w_n$
- ❖ แต่ละชิ้นมีมูลค่า :  $v_1, v_2, v_3, \dots, v_n$
- ❖ ถุงเป้หนึ่งใบจะของได้นักไม่เกิน  $W$
- ❖ ปัญหา : จงเลือกของใส่ถุง เพื่อให้
  - ❖ ถุงไม่ขาด
  - ❖ ได้มูลค่ารวมมากสุด

1		2.5Kg. \$400
2		1.4Kg. \$50
3		2.8Kg. \$350
4		0.7Kg. \$150
		4Kg.

# 0/1 Knapsack

- ❖ ของ  $n$  ชิ้นมีหมายเลข :  $1, 2, 3, \dots, n$
- ❖ แต่ละชิ้นหนัก :  $w_1, w_2, w_3, \dots, w_n$
- ❖ แต่ละชิ้นมีมูลค่า :  $v_1, v_2, v_3, \dots, v_n$
- ❖ ถุงเป้หนึ่งใบจุของได้หนักไม่เกิน  $W$
- ❖ หา  $\langle x_1, x_2, x_3, \dots, x_n \rangle$ ,  $x_k = 0$  หรือ 1

$$\text{maximize} \sum_{k=1}^n x_k v_k$$

$$\text{subject to} \sum_{k=1}^n x_k w_k \leq W$$

$$x_k \in \{0,1\}$$

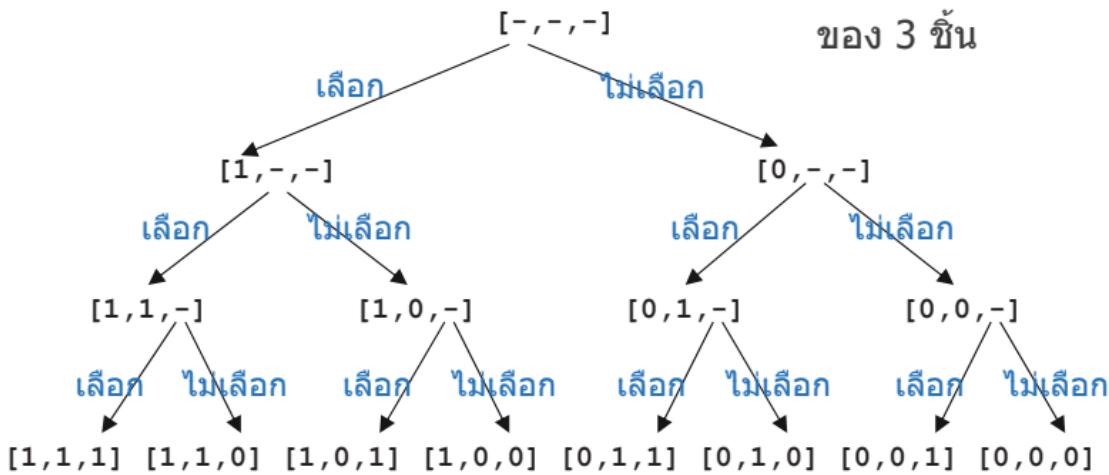
<b>1</b>		2.5Kg. \$400
<b>2</b>		1.4Kg. \$50
<b>3</b>		2.8Kg. \$350
<b>4</b>		0.7Kg. \$150



4Kg.

 $\langle x_1, x_2, x_3, x_4 \rangle$   
 $\langle 1, 0, 0, 1 \rangle$

# 0/1 Knapsack : State Space



# 0/1 Knapsack : ลุยทุกเซตของ

```

knapsack( w[1..n], v[1..n], W ) {
    x = new array[1..n] with all 0's
    vmax = -∞; xmax = x
    knapsack( w, v, W, x, 0 )
    return xmax
}
knapsack(w[1..n], v[1..n], W, x[1..n], m ) {

    if (m == n) {
        if (sum(w,x) <= W AND sum(v,x) > vmax) {
            vmax = sum(v, x); xmax = x
        }
    } else {
        x[m+1] = 1; knapsack( w, v, W, x, m+1 )
        x[m+1] = 0; knapsack( w, v, W, x, m+1 )
    }
}

```

$$\text{sum}(w, x) = \sum_{j=1}^n w[j]x[j]$$

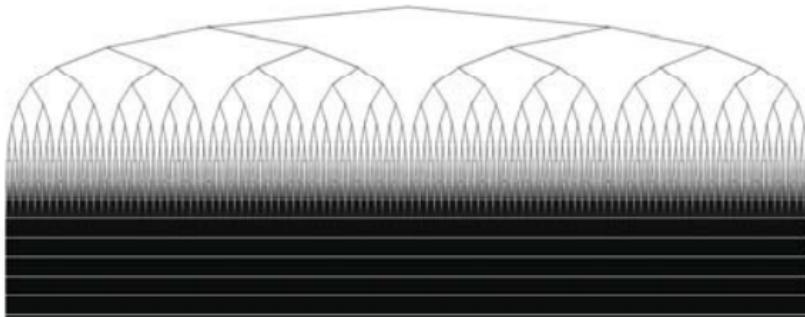
$$\text{sum}(v, x) = \sum_{j=1}^n v[j]x[j]$$

# 0/1 Knapsack : Backtracking

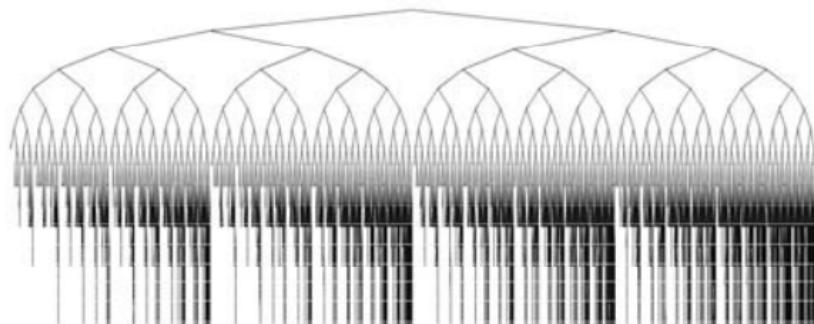
```
knapsack( w[1..n], v[1..n], W ) {  
    x = new array[1..n] with all 0's  
    vmax = -∞; xmax = x  
    knapsack( w, v, W, x, 0 )  
    return xmax  
}  
  
knapsack(w[1..n], v[1..n], W, x[1..n], m) {  
    if (sum(w, x, m) > W) return ←.....เพิ่มเงื่อนไขอะไร  
    if (m == n) {  
        if (sum(w,x) ≤ W AND sum(v,x) > vmax) {  
            vmax = sum(v, x); xmax = x  
        }  
    } else {  
        x[m+1] = 1; knapsack( w, v, W, x, m+1 )  
        x[m+1] = 0; knapsack( w, v, W, x, m+1 )  
    }  
}
```

$$\text{sum}(w, x, m) = \sum_{j=1}^m w[j]x[j]$$

# 0/1 Knapsack : Backtracking



เต็มดัน



เพิ่มการ  
ย้อนรอย

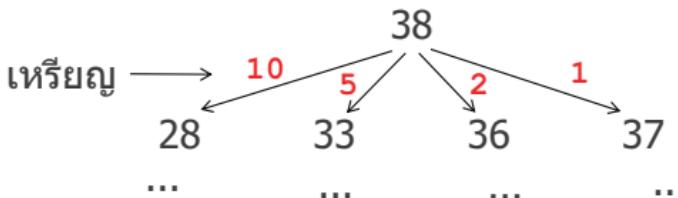
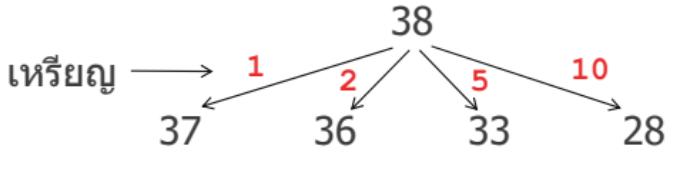
# THONBURI

- ❖ มีเหรียญอยู่  $n$  แบบ :  $1, 2, 3, \dots, n$
- ❖ แต่ละแบบมีค่า :  $v_1, v_2, v_3, \dots, v_n$
- ❖ ต้องการทอนเงินให้ลูกค้ามูลค่า  $V$
- ❖ โดยใช้เหรียญเป็นจำนวนน้อยสุด
  - ❖ (มีเหรียญแต่ละแบบจำนวนไม่จำกัด)
- ❖ ตัวอย่าง : สกุลเงินไทย
  - ❖ มีเหรียญ : 1, 2, 5, 10 บาท (ขอไม่ใช้เหรียญสิบ)
  - ❖ ต้องการทอนเงินมูลค่า 38 บาท
  - ❖ คำตอบ : 10 บาท 3 เหรียญ,  
5 บาท 1 เหรียญ,  
2 บาท 1 เหรียญ,  
1 บาท 1 เหรียญ



# ลองคิดดู

- ❖ เขียนปริภูมิสถานะของการทอนเงิน
- ❖ ควรค้นแบบใด ? (DFS, BFS), ย้อนรอยเมื่อไร ?
- ❖ ลองแบบมีเหรียญ 1, 2, 5, 10 กับ 1, 4, 6, 10

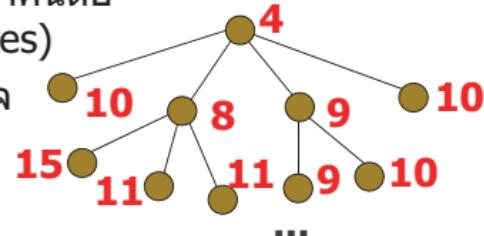




# Least-Cost Search

# Least-Cost Search

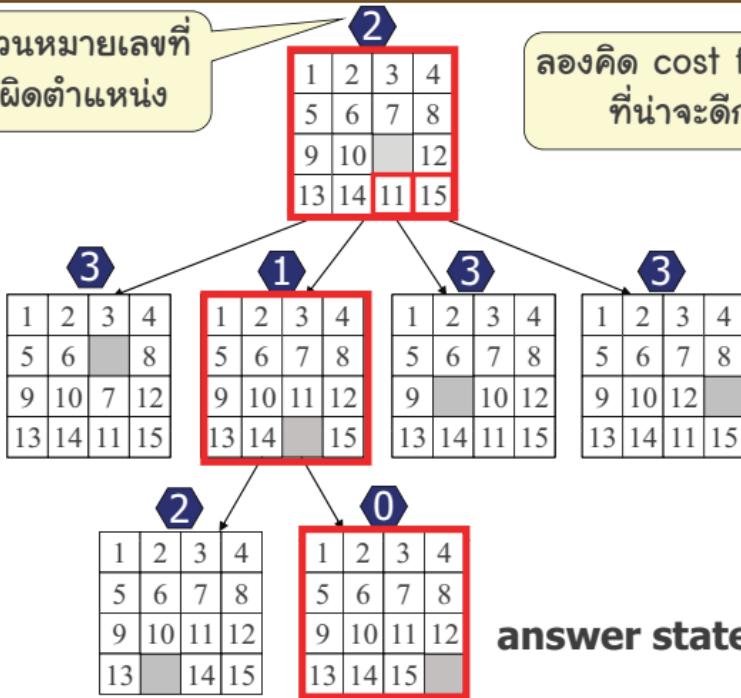
- ❖ ลำดับการเลือก state เพื่อผลิต state ในมีของ DFS และ BFS เป็นไปอย่างมีระเบียบ
- ❖ แต่ไร้เหตุผล !!!
- ❖ blind/uninformed search
- ❖ Least-cost search
  - ❖ แต่ละ state มีต้นทุน (cost function) ในการค้นเพื่อให้ได้คำตอบ
  - ❖ เลือก state ที่มีต้นทุนต่ำสุดมาค้นต่อ (ใช้ priority queue เก็บ states)
  - ❖ ใช้ต้นทุนต่ำสุดเป็นตัวตัดสินใจ (คล้าย greedy แต่ไปแล้ว ไม่ตี กลับใจได้)



# 15-Puzzle : Least-Cost Search

จำนวนหมายเลขอี้ยง  
อยู่ผิดตำแหน่ง

ลองคิด cost function  
ที่น่าจะดีกว่านี้

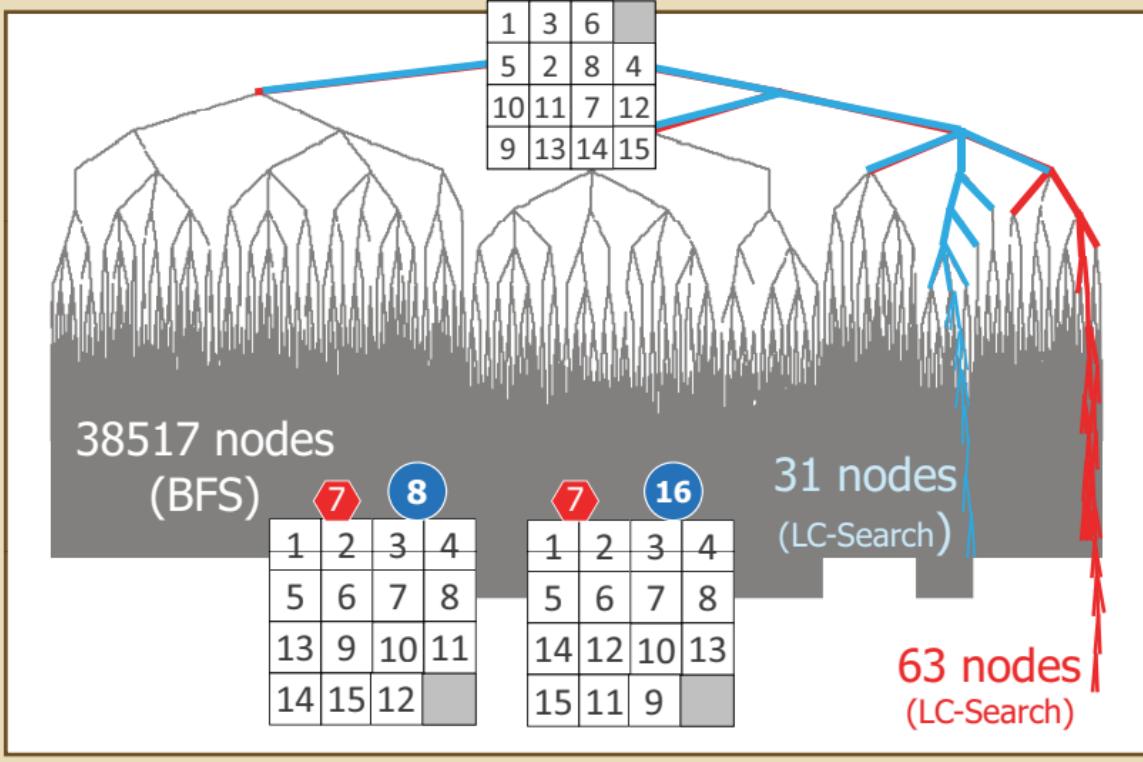


# 15-puzzle : LC-Search

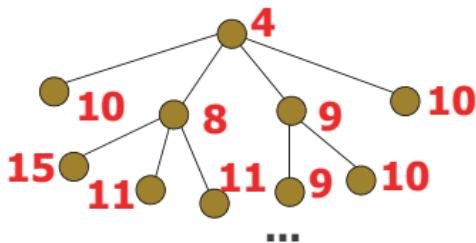
```
solve( board ) {  
    set = an empty set  
    PQ = an empty priority queue  
    PQ.add( board );      set.add( board )  
    while ( PQ ≠ Ø ) {  
        b = PQ.removeMin()  
        if ( b.isAnswer() ) return b;  
        for each d ∈ {UP, DOWN, LEFT, RIGHT} {  
            if ( b.canMoveBlank( d ) ) {  
                b1 = b.moveBlank( d );  
                if (b1 ∉ set) {  
                    PQ.add( b1);      set.add( b1 )  
                }  
            }  
        }  
    }  
    print "no solution";  
}
```

priority queue จัดลำดับ  
ตารางตาม cost function

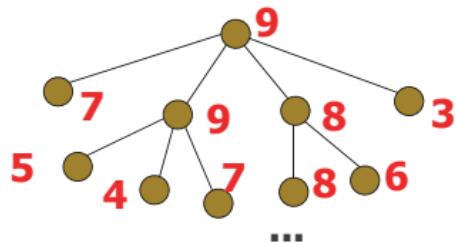
# Breadth-First vs. Least-Cost



# Least-Cost & Max-Profit



min heap



max heap

Best-First Search

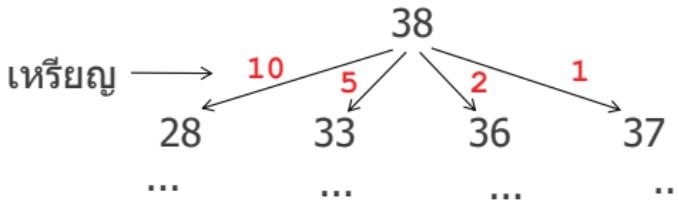
# THONGEN

- ❖ มีเหรียญอยู่  $n$  แบบ :  $1, 2, 3, \dots, n$
- ❖ แต่ละแบบมีค่า :  $v_1, v_2, v_3, \dots, v_n$
- ❖ ต้องการทอนเงินให้ลูกค้ามูลค่า  $V$
- ❖ โดยใช้เหรียญเป็นจำนวนน้อยสุด
  - ❖ (มีเหรียญแต่ละแบบจำนวนไม่จำกัด)
- ❖ ตัวอย่าง : สกุลเงินไทย
  - ❖ มีเหรียญ : 1, 2, 5, 10 บาท (ขอไม่ใช้เหรียญสลึง)
  - ❖ ต้องการทอนเงินมูลค่า 38 บาท
  - ❖ คำตอบ : 10 บาท 3 เหรียญ,  
5 บาท 1 เหรียญ,  
2 บาท 1 เหรียญ,  
1 บาท 1 เหรียญ



## ลองคิดดู : การทอนเงิน

- ❖ จากปริภูมิสถานะของการทอนเงิน 38 ด้วยเหรียญ 1, 2, 5, 10 ข้างล่างนี้



- ❖ จงออกแบบ cost ที่น่าใช้กับ LC-search
- ❖ แล้วลองใช้ค้นหาการทอนเงิน 41 ด้วยเหรียญ 1, 5, 10, 12, 25 ดู

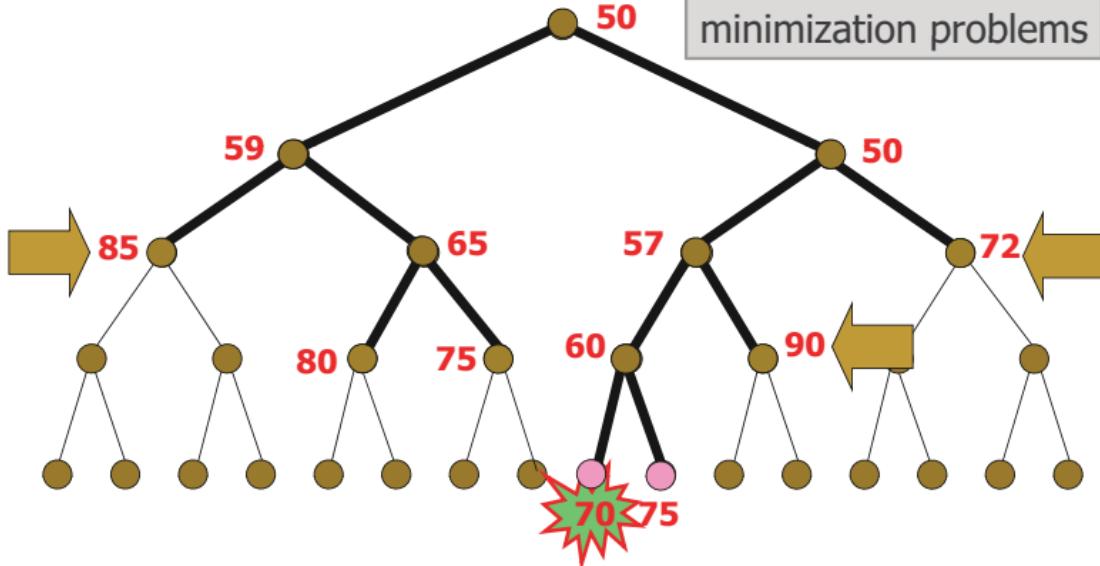
# Branch-and-Bound

# Branch & Bound

- ❖ เหมาะกับ optimization problems
- ❖ Maximization :
  - ❖ แต่ละปมมี upper bound ของค่าที่ต้องการ maximize
  - ❖ Knapsack : [1, 0, 1, -, -, -, -] ให้มูลค่ารวมได้ไม่เกิน 235
- ❖ Minimization :
  - ❖ แต่ละปมมี lower bound ของค่าที่ต้องการ minimize
  - ❖ TSP : [1, 4, 3, -, -, -] ให้ระยะทางรวมอย่างน้อย 213
- ❖ มักใช้ bound ข้างต้นเป็นตัวนำทางในการค้นด้วย

# Least-Cost Search + Bound

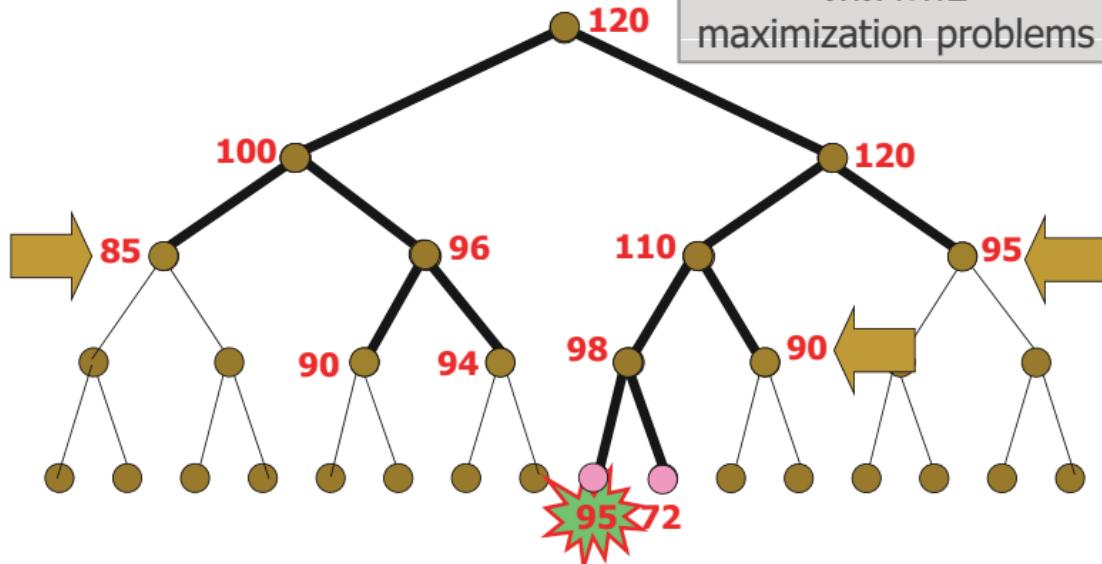
เหมาะกับ  
minimization problems



lower bound ที่ น้อยสุด ตอนนี้มีค่าไม่น้อยกว่า คำตอบดีสุด ที่รู้มา  
ก็แสดงว่า พบรคำตอบที่น้อยสุดแล้ว

# Max-Profit Search + Bound

## หมายเหตุ maximization problems



upper bound ที่มากสุดตอนนี้มีค่าไม่มากกว่าคำตอบดีสุดที่รู้มากแสดงว่า พบคำตอบที่มากสุดแล้ว

# 0/1 Knapsack

- ❖ ของ  $n$  ชิ้นมีหมายเลข :  $1, 2, 3, \dots, n$
- ❖ แต่ละชิ้นหนัก :  $w_1, w_2, w_3, \dots, w_n$
- ❖ แต่ละชิ้นมีมูลค่า :  $v_1, v_2, v_3, \dots, v_n$
- ❖ ถุงเป้หนึ่งใบจุของได้หนักไม่เกิน  $W$
- ❖ หา  $\langle x_1, x_2, x_3, \dots, x_n \rangle$ ,  $x_k = 0$  หรือ 1

$$\text{maximize} \sum_{k=1}^n x_k v_k$$

$$\text{subject to} \sum_{k=1}^n x_k w_k \leq W$$

maximization  
problems

$$x_k \in \{0,1\}$$

<b>1</b>		2.5Kg. \$400
<b>2</b>		1.4Kg. \$50
<b>3</b>		2.8Kg. \$350
<b>4</b>		0.7Kg. \$150
		4Kg.

$$\begin{aligned} & \langle x_1, x_2, x_3, x_4 \rangle \\ & \langle 1, 0, 0, 1 \rangle \end{aligned}$$

# Fractional Knapsack

- ❖ ถ้าอนุญาตให้ "เนื่อง" ของบางส่วนใส่ถุงได้
  - ❖  $0 \leq x_k \leq 1$ , ได้มูลค่าเป็นสัดส่วนตามปริมาณที่เนื่อง
- ❖ สามารถหาคำตอบได้รวดเร็ว (Greedy algo.)
  - ❖ ไล่หยอดเริ่มจากของที่มีมูลค่าต่อน้ำหนักมากสุดใส่ถุงไปเรื่อย ๆ จนถึงขั้นที่ถ้าใส่ ถุงจะขาด ก็เนื่องของใส่ถุงให้พอดีที่ถุงรับน้ำหนักได้
- ❖ ถ้าของ  $1, 2, \dots, n$  เรียงแล้ว  $\frac{v_1}{w_1} \geq \frac{v_2}{w_2} \geq \dots \geq \frac{v_n}{w_n}$
- ❖ ใช้เวลา  $O(n)$

$$x_1=1.0 \quad x_2=1.0 \quad x_3=1.0 \quad x_4=0.8$$

$$v_1=66, \quad v_2=20, \quad v_3=30, \quad v_4=60, \quad v_5=40 \quad \sum v = 164$$

$$w_1=30, \quad w_2=10, \quad w_3=20, \quad w_4=50, \quad w_5=40 \quad W = 100$$

$$v/w : 2.2 \quad 2.0 \quad 1.5 \quad 1.2 \quad 1.0$$

# Upper Bound ของ 0/1 Knapsack

$$v_1=66, v_2=20, v_3=30, v_4=60, v_5=40$$

$$W = 100$$

$$w_1=30, w_2=10, w_3=20, w_4=50, w_5=40$$

มูลค่ารวมของ  
คำตอบของ  
0/1 Knapsack



มูลค่ารวมของ  
คำตอบของ  
Fractional Knapsack

$$<1, 1, 1, 0, 1>$$

$$\Sigma v = \textcolor{red}{156}$$

$$<1, 1, 1, 0.8, 0>$$

$$\Sigma v = \textcolor{red}{164}$$

# Upper Bound ของมูลค่ารวม

- ❖ มีของอยู่  $n$  ชิ้น
- ❖ เรียงชิ้นที่  $1, 2, \dots, n$  ตาม  $v/w$  จากมากไปน้อย
- ❖ คำตอบ  $x_1, x_2, \dots, x_{m+1}$   
 $1, 0 \dots 1,$   
ตัดสินใจแล้วว่าจะเลือกหรือไม่เลือก  
 $\sum x_i v_i$
- ❖  $x_{m+2} \dots, x_n$   
?, ?, ..., ?  
ยังไม่ตัดสินใจ  
???
- ❖ สามารถคำนวณ  $\max \sum v$  ในส่วนที่ยังไม่ตัดสินใจด้วยการหาคำตอบของ **fractional knapsack** ตั้งแต่ชิ้นที่  $m+2$  ถึง  $n$  เมื่อถุงจุได้  $W - \sum x_i w_i$

# ตัวอย่าง

$$W = 100$$

~~$v_1=66$~~ ,  
 ~~$w_1=30$~~

$v_2=20$ ,  $v_3=30$ ,  $v_4=60$ ,  $v_5=40$   
 $w_2=10$ ,  $w_3=20$ ,  $w_4=50$ ,  $w_5=40$

$$v/w : 2.2 \quad 2.0 \quad 1.5 \quad 1.2 \quad 1.0$$

$$\Sigma v = 0 + 20 + 30 + 60 + 20 = 130$$

$$W = 100 - 30 = 70$$

~~$v_1=66$~~ ,  
 ~~$w_1=30$~~

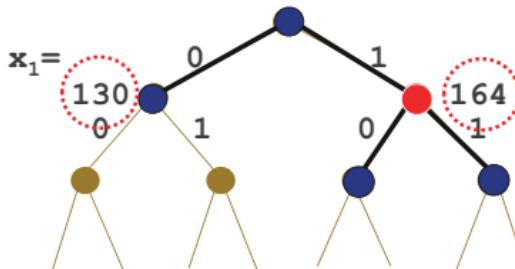
$v_2=20$ ,  $v_3=30$ ,  $v_4=60$ ,  $v_5=40$   
 $w_2=10$ ,  $w_3=20$ ,  $w_4=50$ ,  $w_5=40$

$$v/w : 2.2 \quad 2.0 \quad 1.5 \quad 1.2 \quad 1.0$$

$$\Sigma v = 66 + 20 + 30 + 48 + 0 = 164$$

# 0/1 Knapsack

❖ Max-Profit search เลือกแต่ก็ปมที่กำไรสูงสุด



$$x_1=0, W=100-0 \\ \max (\Sigma v \text{ ที่เหลือ}) \text{ เมื่อ}$$

$$x_2=1, x_3=1, x_4=1, x_5=0.5 \\ \text{ได้ } \max \Sigma v = 0 + 130$$

$$x_1=1, W=100-30=70 \\ \max (\Sigma v \text{ ที่เหลือ}) \text{ เมื่อ} \\ x_2=1, x_3=1, x_4=0.8, x_5=0 \\ \text{ได้ } \max \Sigma v = 66 + 98 = 164$$

$$W = 100$$

~~$v_1=60, w_1=30$~~

$v_2=20, v_3=30, v_4=60, v_5=40$

$w_2=10, w_3=20, w_4=50, w_5=40$

$v/w : 2.2$

$2.0$

$1.5$

$1.2$

$1.0$

# 0/1 Knapsack : Branch & Bound

```

Knapsack( w[1..n], v[1..n], W ) { // sorted by v/w
    H = an empty max heap; VMAX = -∞
    H.add([], FKS(w,v,W,1,n))
    while( H ≠ ∅ ) {
        (x[1..m], VUB) = H.removeMax()
        if (VUB ≤ VMAX) break; → BOUND
        if (m == n) {
            if (VUB > VMAX) { VMAX = VUB; xMAX = x }
        } else {
            Σv = sum(v, x, m); Σw = sum(w, x, m)
            x0 = copyOf(x, m+1); x0[m+1] = 0
            H.add(x0, 0+Σv + FKS(w, v ,W - Σw, m+2, n))
            if (W - Σw - w[m+1] ≥ 0) {
                x1 = copyOf(x, m+1); x1[m+1] = 1
                H.add(x1, v[m+1]+Σv +FKS(w,v,W-Σw-w[m+1],m+2,n))
            }
        }
    }
    return xMAX
}

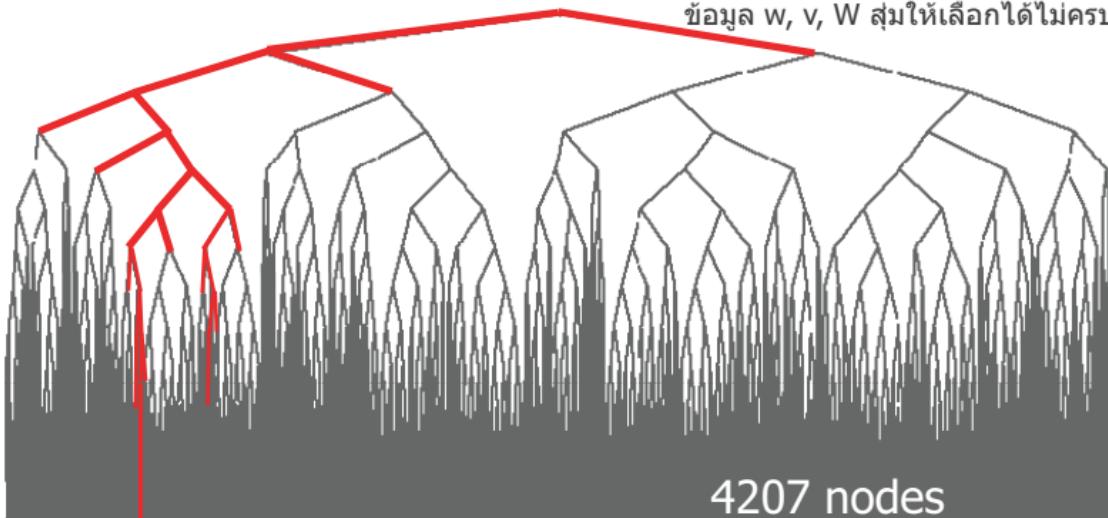
```

BOUND

BRANCH

# BT & BB

Knapsack :  $n = 13$ ,  
ข้อมูล  $p, v, W$  สุ่มให้เลือกได้ไม่ครบ



34 nodes  
(Branch & Bound)

## การออกรูปแบบอัลกอริทึม

<b>n</b>	<b>BFS</b>	<b>Backtrack</b>	<b>B&amp;B</b>	<b>n</b>	<b>B&amp;B</b>
1	3	3	3	50	86
2	7	5	5	100	188
3	15	6	6	150	232
4	31	12	10	200	308
5	63	33	11	250	423
6	127	70	12	300	555
7	255	120	12	350	564
8	511	148	15	400	609
9	1,023	302	18	450	775
10	2,047	944	18	500	781
11	4,095	1,829	23	550	998
12	8,191	4,996	20	600	991
13	16,383	4,207	34	650	1,088
14	32,767	11,562	39	700	1,159
15	65,535	17,587	22	750	1,312
16	131,071	55,583	28	800	1,351
17	262,143	119,443	47	850	1,488
18	524,287	68,944	34	900	1,334
19	1,048,575	694,689	39	950	1,576
20	2,097,151	288,154	29	1,000	1,699

จำนวน  
ปัมที่ค้น

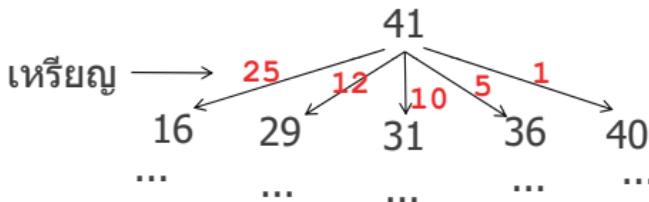
# THONBON

- ❖ มีเหรียญอยู่  $n$  แบบ :  $1, 2, 3, \dots, n$
- ❖ แต่ละแบบมีค่า :  $v_1, v_2, v_3, \dots, v_n$
- ❖ ต้องการทอนเงินให้ลูกค้ามูลค่า  $V$
- ❖ โดยใช้เหรียญเป็นจำนวนน้อยสุด
  - ❖ มีเหรียญแต่ละแบบจำนวนไม่จำกัด
- ❖ ตัวอย่าง
  - ❖ มีเหรียญ : 1, 5, 10, 12, 25 บาท
  - ❖ ต้องการทอนเงินมูลค่า 41 บาท
  - ❖ คำตอบ : 12 บาท 3 เหรียญ,  
5 บาท 1 เหรียญ



# การทอนเงิน : branch & bound

- ❖ จากปริภูมิสถานะของการทอนเงิน 41 ด้วยเหรียญ 1, 5, 10, 12, 25 ข้างล่างนี้

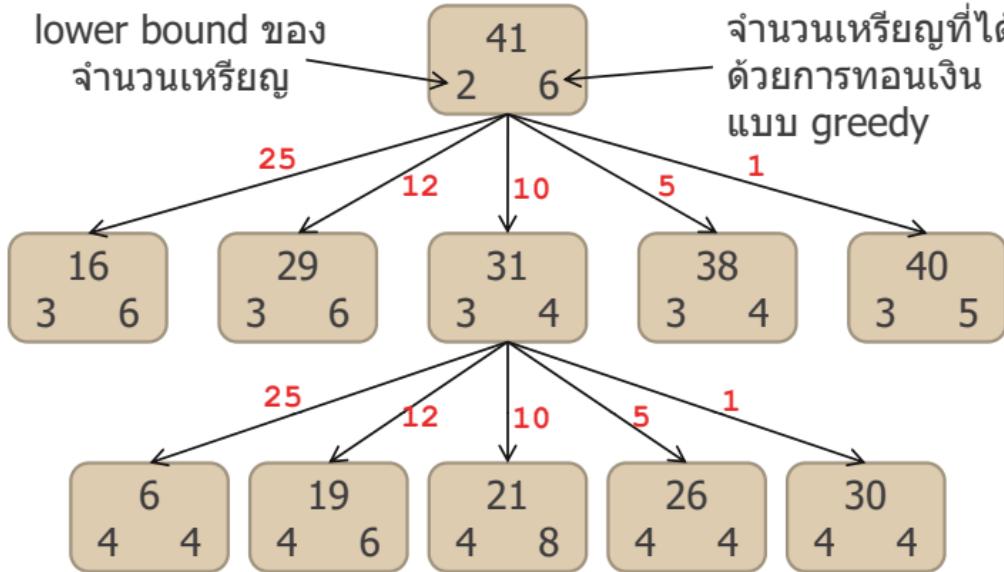


- ❖ ใช้ผลการทอนเหรียญแบบ greedy เป็น cost (ทอนด้วยเหรียญที่มีค่ามากสุดก่อน) ช่วยนำทาง
- ❖ lower bound ของจำนวนเหรียญที่ปั๊ม x คือ
  - ❖ จำนวนเหรียญที่ทอนมาจนถึงปั๊ม x บวกกับ
 
$$\left\lceil \frac{\text{เงินที่ยังไม่ทอนที่ปั๊ม } x}{\text{มูลค่าเหรียญมากสุดที่น้อยกว่าเงินที่ยังไม่ทอน}} \right\rceil$$

# การทอนเงิน : branch & bound

lower bound ของ  
จำนวนเหรียญ

จำนวนเหรียญที่ได้  
ด้วยการทอนเงิน  
แบบ greedy



ลองว่าดูต่อจะนัดคำตอบ

# การแจงผลเฉลยแบบ เรียงสับเปลี่ยน

# การแจงผลเฉลย : permutation

❖ หลายปัญหามีผลเฉลยเป็นแบบการเรียงสับเปลี่ยน

- ❖ n-Queen
- ❖ assignment problem
- ❖ travelling salesman problem
- ❖ bandwidth minimization problem
- ❖ ...

[1,2,3,4]	[2,1,3,4]	[3,2,1,4]	[4,2,3,1]
[1,2,4,3]	[2,1,4,3]	[3,2,4,1]	[4,2,1,3]
[1,3,2,4]	[2,3,1,4]	[3,1,2,4]	[4,3,2,1]
[1,3,4,2]	[2,3,4,1]	[3,1,4,2]	[4,3,1,2]
[1,4,3,2]	[2,4,3,1]	[3,4,1,2]	[4,1,3,2]
[1,4,2,3]	[2,4,1,3]	[3,4,2,1]	[4,1,2,3]

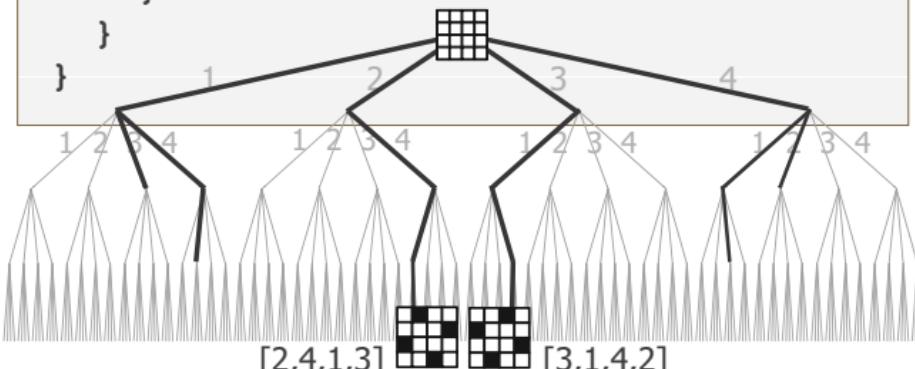
$n$	$n!$
1	1
2	2
3	6
4	24
5	120
6	720
7	5,040
8	40,320
9	362,880
10	3,628,800
11	39,916,800
12	479,001,600

# n-Queen : Backtracking

```

queen(col[1..n], m) {
    if (m == n) print(col)
    else {
        for (i = 1; i <= n; i++) {
            col[m+1] = i;
            if (isValid(col, m+1))
                queen(col, m + 1);
        }
    }
}

```



[1,1,1,1]  
[1,1,1,2]  
[1,1,1,3]  
[1,1,1,4]  
[1,1,2,1]  
[1,1,2,2]  
[1,1,2,3]  
[1,1,2,4]

...

[4,4,3,1]  
[4,4,3,2]  
[4,4,3,3]  
[4,4,3,4]  
[4,4,4,1]  
[4,4,4,2]  
[4,4,4,3]  
[4,4,4,4]

# ปรับปรุงให้ปริภูมิเล็กลง

ไม่มีกรณีว่างค้วนใน  
คอลัมน์เดียวกัน

$$4! = 24 \text{ กรณี}$$

- [1,2,3,4]
- [1,2,4,3]
- [1,3,2,4]
- [1,3,4,2]
- [1,4,3,2]
- [1,4,2,3]
- [2,1,3,4]
- [2,1,4,3]

- ...
- [3,4,1,2]
- [3,4,2,1]
- [4,2,3,1]
- [4,2,1,3]
- [4,3,2,1]
- [4,3,1,2]
- [4,1,3,2]
- [4,1,2,3]

ต้องตรวจกรณีว่าง  
ค้วนในคอลัมน์  
เดียวกัน

1		
1		
	2	
		4

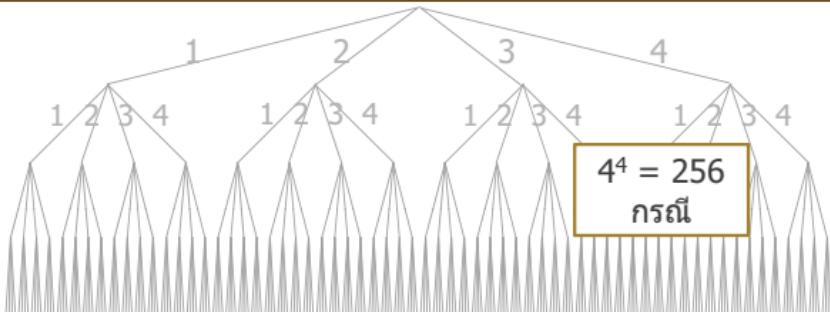
$$4^4 = 256 \text{ กรณี}$$

- [1,1,1,1]
- [1,1,1,2]
- [1,1,1,3]
- [1,1,1,4]
- [1,1,2,1]
- [1,1,2,2]
- [1,1,2,3]
- [1,1,2,4]

- ...
- [4,4,3,1]
- [4,4,3,2]
- [4,4,3,3]
- [4,4,3,4]
- [4,4,4,1]
- [4,4,4,2]
- [4,4,4,3]
- [4,4,4,4]

# ปรับปรุงให้ปริภูมิเล็กลง

[1,2,3,4]  
 [1,2,4,3]  
 [1,3,2,4]  
 [1,3,4,2]  
 [1,4,3,2]  
 [1,4,2,3]  
 [2,1,3,4]  
 [2,1,4,3]

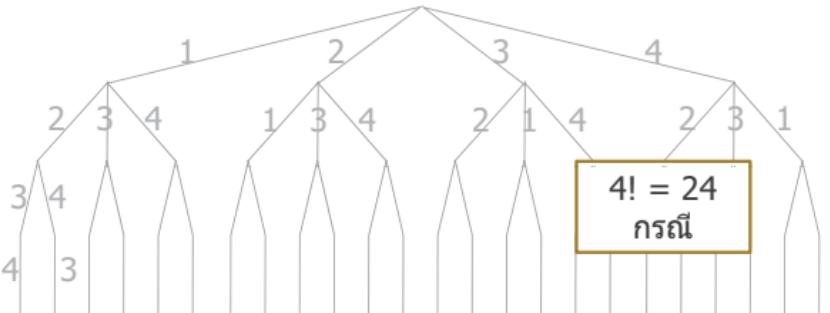


$$4^4 = 256$$

กรณี

[1,1,1,1]  
 [1,1,1,2]  
 [1,1,1,3]  
 [1,1,1,4]  
 [1,1,2,1]  
 [1,1,2,2]  
 [1,1,2,3]  
 [1,1,2,4]

[3,4,1,2]  
 [3,4,2,1]  
 [4,2,3,1]  
 [4,2,1,3]  
 [4,3,2,1]  
 [4,3,1,2]  
 [4,1,3,2]  
 [4,1,2,3]

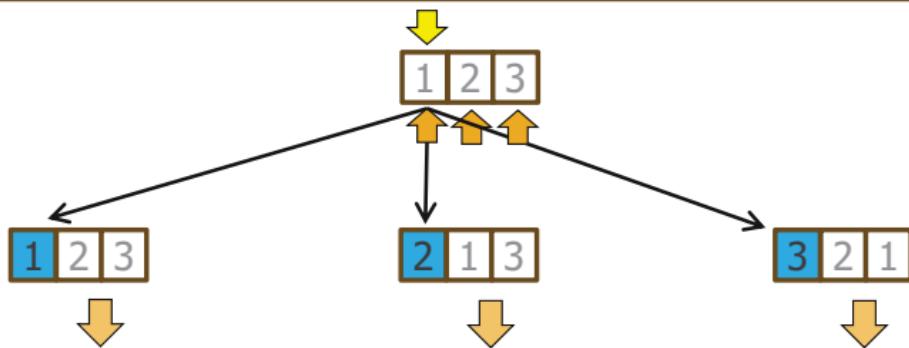


$$4! = 24$$

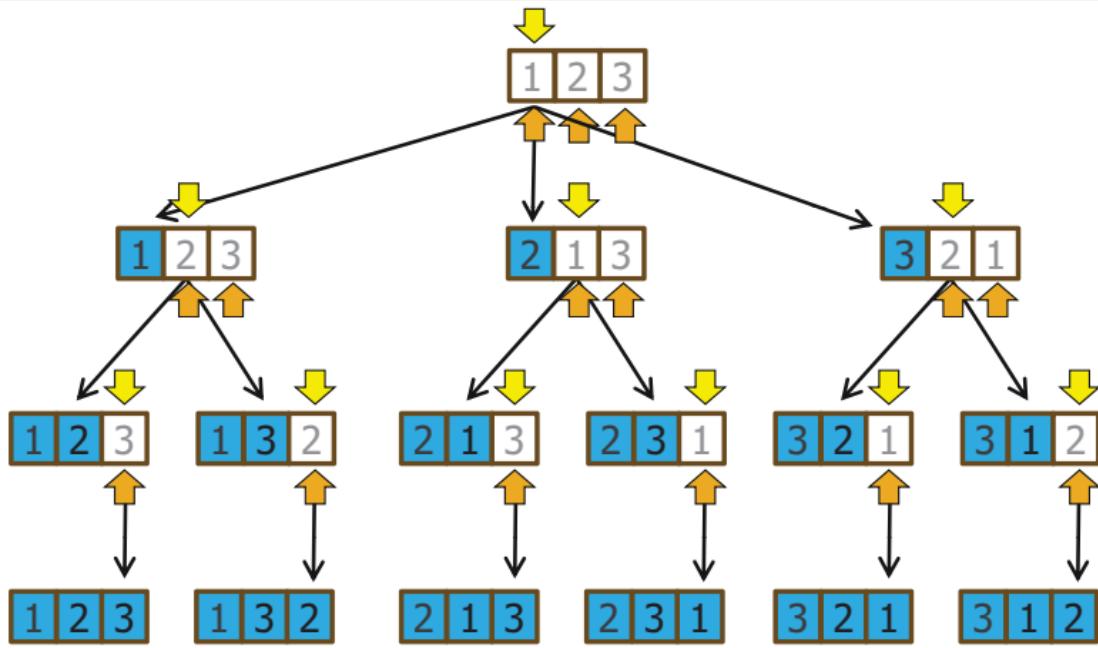
กรณี

[4,4,3,1]  
 [4,4,3,2]  
 [4,4,3,3]  
 [4,4,3,4]  
 [4,4,4,1]  
 [4,4,4,2]  
 [4,4,4,3]  
 [4,4,4,4]

# การแจกแจงการเรียงสับเปลี่ยน

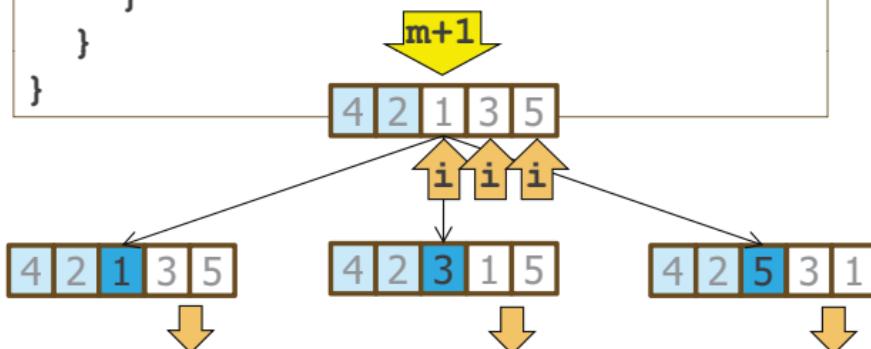


# การแจกแจงการเรียงลำดับเปลี่ยน



# การแจกแจงการเรียงสับเปลี่ยน

```
perm( x[1..n], m ) {  
    if (n == m) print( x )  
    else {  
        for(i = m+1; i <= n; i++) {  
            x[i] ↔ x[m+1]  
            perm(x, m+1)  
            x[i] ↔ x[m+1]  
        }  
    }  
}
```

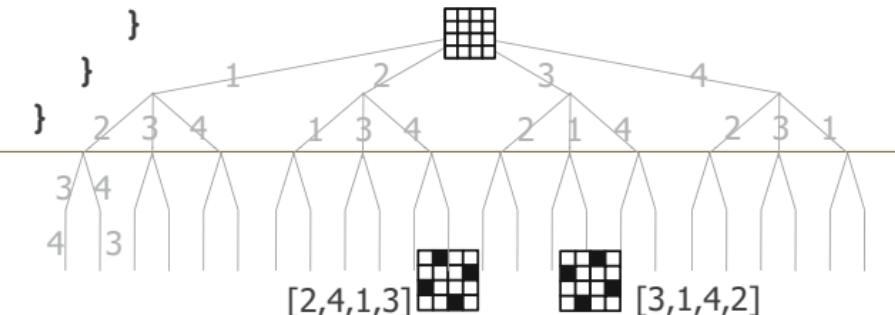


# n-Queen : DFS

```

queen(x[1..n], m) {
    if (n == m)
        if (isValid(x)) print( x )
    else {
        for(i = m+1; i <= n; i++) {
            x[i] ↔ x[m+1]
            queen(x, m+1)
            x[i] ↔ x[m+1]
        }
    }
}

```



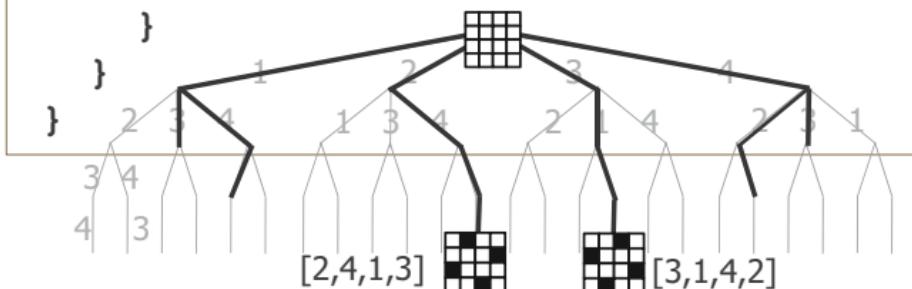
[1,2,3,4]  
[1,2,4,3]  
[1,3,2,4]  
[1,3,4,2]  
[1,4,3,2]  
[1,4,2,3]  
[2,1,3,4]  
[2,1,4,3]

...

[3,4,1,2]  
[3,4,2,1]  
[4,2,3,1]  
[4,2,1,3]  
[4,3,2,1]  
[4,3,1,2]  
[4,1,3,2]  
[4,1,2,3]

## n-Queen : Backtracking

```
queen(x[1..n], m) {  
    if (n == m)  
        if (isValid(x)) print( x )  
    else {  
        for(i = m+1; i <= n; i++) {  
            x[i] ↔ x[m+1]  
            if (isValid(x, m+1))  
                queen(x, m+1)  
            x[i] ↔ x[m+1]
```



[1,2,3,4]  
[1,2,4,3]  
[1,3,2,4]  
[1,3,4,2]  
[1,4,3,2]  
[1,4,2,3]  
[2,1,3,4]  
[2,1,4,3]

—

[3,4,1,2]  
 [3,4,2,1]  
 [4,2,3,1]  
 [4,2,1,3]  
 [4,3,2,1]  
 [4,3,1,2]  
 [4,1,3,2]  
 [4,1,2,3]

# $n$ -Queen : Backtracking

$n$	จำนวน ค่าตอบ	จำนวนการเรียกข้า จนพบค่าตอบแรก		เวลาในการหา ทุกค่าตอบ (วินาที)	
		แบบแรก ( $n^n$ )	แบบที่สอง ( $n!$ )	แบบแรก ( $n^n$ )	แบบที่สอง ( $n!$ )
4	2	9	9	≈ 0	≈ 0
5	10	6	6	≈ 0	≈ 0
6	4	32	33	≈ 0	≈ 0
7	40	10	10	≈ 0	≈ 0
8	92	114	117	≈ 0	≈ 0
9	352	42	44	≈ 0	≈ 0
10	724	103	103	0.01	≈ 0
11	2,680	53	53	0.06	0.05
12	14,200	262	262	0.37	0.26
13	73,712	112	108	2.26	1.55
14	365,596	1,900	1,915	15.13	10.00
15	2,279,184	1,360	1,359	106.26	68.23
16	14,772,512	10,053	10,053	798.30	497.55
17	95,815,104	5,375	5,447	6,213.57	3,747.33

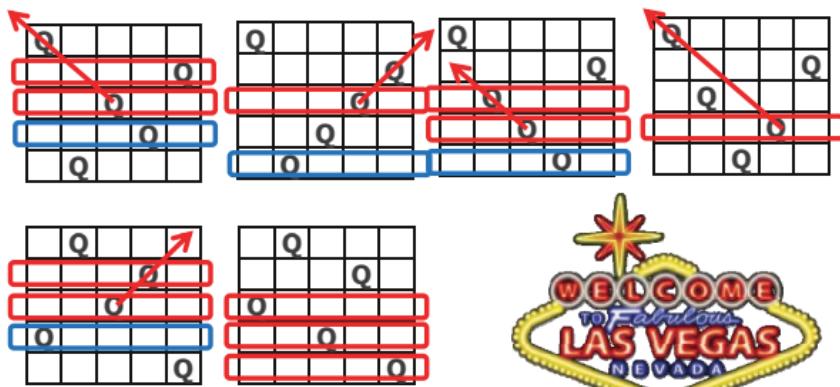
# n-Queen : Backtracking

n	ปริภูมิแบบ permutation (หยุดค้นเมื่อพบคำตอบแรก)	
	จำนวนปม	เวลา (วินาที)
5	6	≈ 0
10	103	≈ 0
15	1,359	0.004
20	199,638	0.073
25	48,652	0.023
30	16,918,415	10.231
35	255,491,565	189.845

นานไป

# n-Queen : Las Vegas

- ❖ เริ่มด้วย permutation สุ่ม ๆ ของ  $1, 2, \dots, n$
- ❖ ตรวจทีละແກว่า เห็นควีนແກวน ๆ หรือไม่
- ❖ ถ้าเห็น ให้สลับควีนของແກวนนั้นกับແກาถัดไป
- ❖ ถ้าສลับครบทุกແກา แล้วยังไม่สำเร็จ ทำใหม่ทั้งหมด



# n-Queen : Las Vegas

```
queenLV( col[1..n] ) {
    for (i = 1; i <= n; i++) col[i] = i;
    while ( queenLV1(col) == false ) {}
    print( col )
}
queenLV1( col[1..n] ) {
    shuffle( col )
    for(k = 2; k <= n; k++) {
        j = k + 1
        while (!isValid(col, k)) {
            if (j > n) return false
            col[k] ↔ col[j];
            j++
        }
    }
    return true
}
```

# n-Queen : Las Vegas

<i>n</i>	ปริภูมิแบบ permutation (หยุดค้นเมื่อพบคำตอบแรก)	
	จำนวนสถานะ	เวลา (วินาที)
100	122,468	0.05
150	110,397	0.06
200	633,492	0.45
250	869,546	0.74
300	5,743,536	5.76
350	3,546,165	4.08
400	10,838,971	14.14
450	15,321,788	22.24
500	4,779,326	7.78
550	126,438,286	218.59
600	12,246,381	23.09
650	156,425,660	315.30
700	76,669,563	164.98
750	299,588,148	696.40
800	134,631,279	333.67

<i>n</i>	จำนวน คำตอบ
4	2
5	10
6	4
7	40
8	92
9	352
10	724
11	2,680
12	14,200
13	73,712
14	365,596
15	2,279,184
16	14,772,512
17	95,815,104

# Assignment Problem

- ❖ มีงาน  $n$  งาน, มีคน  $n$  คน
- ❖ ต้นทุนการให้คนที่  $i$  ทำงานที่  $j$  เท่ากับ  $c_{ij}$
- ❖ จงหาการกำหนดงานให้กับทุกคน ที่ใช้ต้นทุนต่ำสุด (แต่ละคนทำงานต่างกัน)
- ❖ ผลเฉลยอยู่ในรูปแบบ permutation
  - ❖  $(4, 2, 1, 3)$

Branch & Bound

		งาน			
		1	2	3	4
คน	<i>a</i>	14	12	15	20
	<i>b</i>	15	10	14	25
	<i>c</i>	10	19	17	23
	<i>d</i>	16	18	8	40

# Lower Bound ของต้นทุนการให้งาน

- ❖ ใช้อะไรแทน cost นำทางในการค้น (branch) ?
- ❖ เป็น minimization problem
  - ❖ ต้องหา lower bound ของต้นทุนการให้งานกับคน
  - ❖ ต้องจำต้นทุนรวมน้อยสุดของคำตอบที่ดีสุดที่รู้มา
- ❖ คำนวณ lower bound ของต้นทุนการให้งาน ?
  - ❖ ต้นทุนการให้งานที่ตัดสินใจไปแล้ว บวกกับ 15
  - ❖ ต้นทุนน้อยสุดของการให้งานกับคนที่เหลือ  $10 + 10 + 14$

	1	2	3	4
a	18	12	15	20
b	15	10	14	25
c	10	16	17	23
d	14	18	8	40

49

[3, -, -, -]

# Assignment : Branch & Bound

	1	2	3	4
a	18	12	15	20
b	15	10	14	25
c	10	12	17	23
d	14	18	8	40

[-, -, -, -]

$$0 + 12 + 10 + 10 + 8 = 40$$

[1, -, -, -]

	1	2	3	4
a	18	12	15	20
b	15	10	14	25
c	10	12	17	23
d	14	18	8	40

$$18 + 10 + 17 + 18 = 63$$

[2, -, -, -]

	1	2	3	4
a	18	12	15	20
b	15	10	14	25
c	10	12	17	23
d	14	18	8	40

$$12 + 14 + 10 + 8 = 44$$

[3, -, -, -]

	1	2	3	4
a	18	12	15	20
b	15	10	14	25
c	10	12	17	23
d	14	18	8	40

$$15 + 10 + 10 + 14 = 49$$

[4, -, -, -]

	1	2	3	4
a	18	12	15	20
b	15	10	14	25
c	10	12	17	23
d	14	18	8	40

$$20 + 10 + 10 + 8 = 48$$

# Assignment : Branch & Bound

	1	2	3	4
1	18	12	15	20
2	15	10	14	25
3	10	12	17	23
4	14	18	8	40

[2, -, -, -]

$$\begin{aligned}12 + 14 + 10 + 8 \\= 44\end{aligned}$$

[2, 1, -, -]

	1	2	3	4
a		12		
b	15		14	25
c	10		17	23
d	14		8	40

$$\begin{aligned}12 + 15 + 17 + 8 \\= 52\end{aligned}$$

[2, 3, -, -]

	1	2	3	4
a		12		
b			14	25
c			17	23
d			8	40

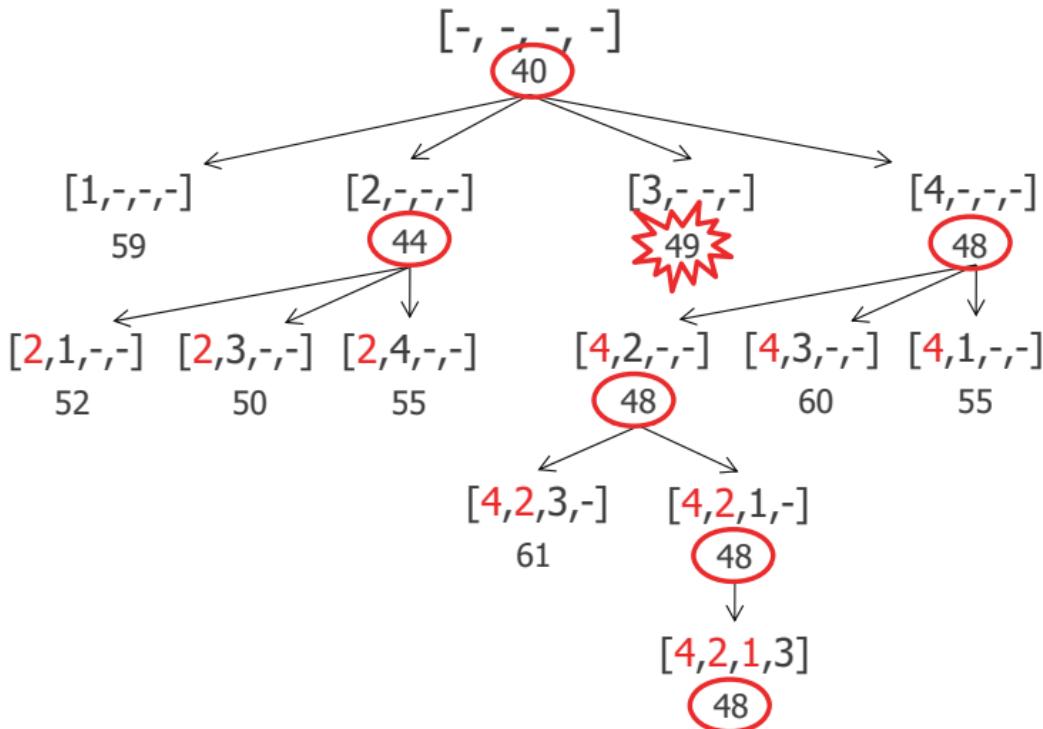
$$\begin{aligned}12 + 14 + 10 + 14 \\= 50\end{aligned}$$

[2, 4, -, -]

	1	2	3	4
a		12		
b			14	25
c			17	23
d			8	40

$$\begin{aligned}12 + 25 + 10 + 8 \\= 55\end{aligned}$$

# Assignment : Branch & Bound



# ควรหาสักหนึ่งคำตอบดอนเริ่มต้น

	1	2	3	4
a	18	12	15	20
b	15	10	14	25
c	10	12	17	23
d	14	18	8	40

หรือ

	1	2	3	4
a	18	12	15	20
b	15	10	14	25
c	10	12	17	23
d	14	18	8	40

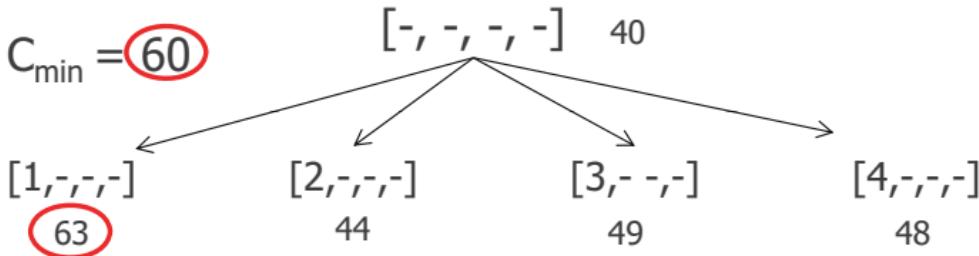
หรือ . . .

$$18+10+17+40 = 85$$

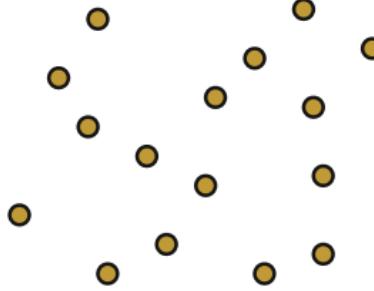
$$20+14+12+14 = 60$$

ให้กับ  $C_{\min}$  ตอนเริ่มต้น

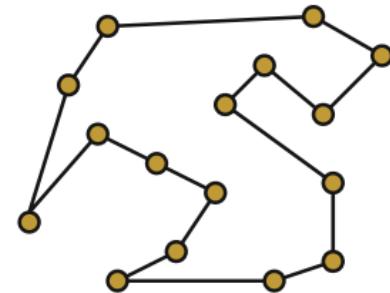
$$C_{\min} = 60$$



# Travelling Salesperson Problem



**Input**



**Output**

## TSP : รูปแบบของผลเฉลย

- ❖ ผ่าน ก เมือง เมืองละครั้ง และกลับมาที่เมืองเริ่มต้น
- ❖ ผลเฉลยอยู่ในรูปแบบ permutation ของ  $1, 2, \dots, n$  (ที่มี 1 เป็นจุดเริ่มต้น เพราะเป็นวงกลับมา 1)
- ❖ ตัวอย่าง :  $n = 4$  มีทั้งหมด  $(4-1)! = 6$ 
  - ❖  $(1, 2, 3, 4), (1, 2, 4, 3), (1, 3, 2, 4),$   
 $(1, 3, 4, 2), (1, 4, 2, 3), (1, 4, 3, 2)$

# TSP : ลุยทุกผลเฉลย (DFS)

```
tsp( g[1..n][1..n] ) {
    LMIN = ∞; xMIN = []
    tsp( g, [1,2,3,...,n], 1 );
    return xMIN
}
tsp( g[1..n][1..n], x[1..n], m ) {
    if (n == m) {
        len = tourLength(g, x)
        if (len < LMIN) {LMIN = len; xMIN = x}
    } else {
        for(i = m+1; i <= n; i++) {
            x[i] ↔ x[m+1]
            tsp( g, x, m+1 )
            x[i] ↔ x[m+1]
        }
    }
}
```

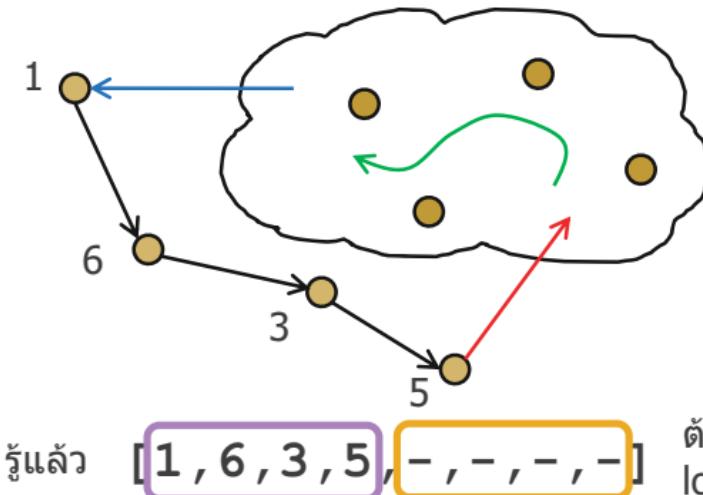
# TSP : คุยทุกผลเฉลย (BFS)

```
tsp( g[1..n] [1..n] ) {
    LMIN = ∞; xMIN = []
    Q = an empty queue
    Q.enqueue([1,2,...,n], 1) {
        while ( Q ≠ ∅ ) {
            (x, m) = Q.dequeue()
            if (n == m) {
                len = tourLength(g, x)
                if (len < LMIN) {LMIN = len; xMIN = x}
            } else {
                for (i = m+1; i ≤ n; i++) {
                    x' = copyOf(x)
                    x'[i] ↔ x'[m+1]
                    Q.enqueue(x', m+1)
                }
            }
        }
    return xMIN
}
```

# TSP : Branch & Bound

## ❖ TSP เป็น minimization problem

- ❖ ต้องหา lower bound ของระยะทางการเดินทาง
- ❖ ต้องจำระยะทางน้อยสุดของการเดินทางที่สั้นสุดที่รู้มา



# TSP : Branch & Bound

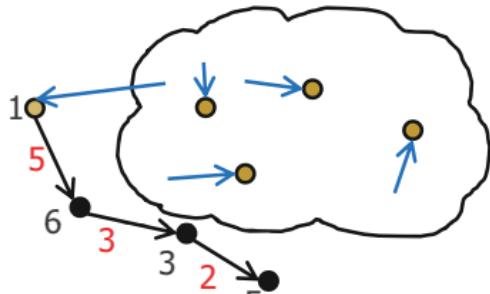
```

tspBB( g[1..n] [1..n] ) {
    H = an empty queue
    xMIN = [1,2,...,n]; LMIN = tourLengthLB(g, xMIN, n)
    H.add(xMIN, 1, tourLengthLB(g, xMIN, 1))
    while ( H ≠ Ø ) {
        (x, m, LLB) = H.removeMin()
        if (LLB ≥ LMIN) break;
        if (n == m) {
            LMIN = LLB; xMIN = x
        } else {
            for (i = m+1; i ≤ n; i++) {
                x' = copyOf(x)
                x'[i] ↔ x'[m+1]
                H.add(x', m+1, tourLengthLB(g, x', m+1))
            }
        }
    }
    return xMIN
}

```

# Lower Bound : $\sum$ สั้นสุดพุ่งเข้า

การเดินทางเป็นวงผ่านทุกปม<sup>ทุกปมต้องมีเส้นเชื่อมพุ่งเข้าหา</sup>



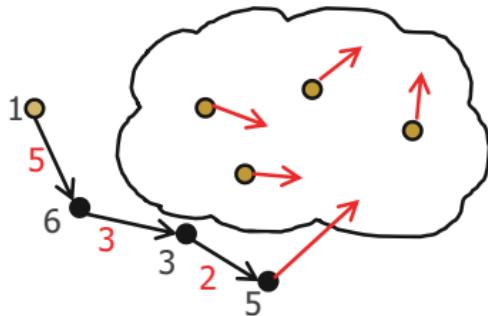
ใช้ผลรวมของความยาว  
เส้นเชื่อมสั้นสุดที่ พุ่งเข้า

1	2	3	4	5	6	7	8	
1	-	9	5	6	3	5	1	2
2	5	-	7	4	7	5	(3)	(2)
3	1	3	-	5	2	1	1	3
4	2	4	1	-	3	7	6	3
5	2	(3)	4	6	-	2	4	3
6	2	4	(3)	5	1	-	6	3
7	(1)	5	3	6	3	7	-	4
8	7	4	5	(3)	6	2	5	-

$$5+3+2 + 1+3+3+3+2 = 22$$

# Lower Bound : $\Sigma$ สั้นสุดพุ่งออก

การเดินทางเป็นวงผ่านทุกปม  
ทุกปมต้องมีเส้นเชื่อม พุงออก



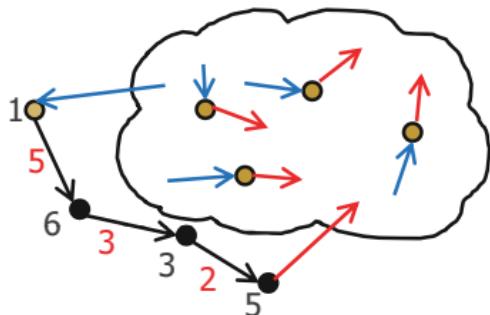
ใช้ผลรวมของความยาว  
เส้นเชื่อม สั้นสุด ที่ พุงออก

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

$$5+3+2 + 2+2+2+1+3 = 20$$

# Lower Bound : $\Sigma$ สั้นสุดพุ่งเข้าออก

การเดินทางเป็นวงผ่านทุกปม  
ทุกปมต้องมีเส้นเชื่อมพุงเข้าและออก



ใช้ผลรวมของความยาว  
ครึ่งหนึ่งเส้นเชื่อมสั้นสุดที่ พุงเข้า  
ครึ่งหนึ่งเส้นเชื่อมสั้นสุดที่ พุงออก

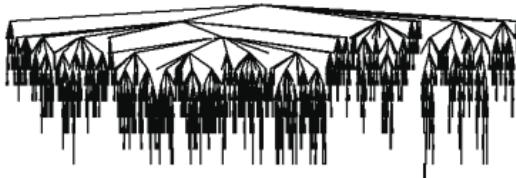
1	2	3	4	5	6	7	8
1	-	5	6	3	5	1	2
2	5	-	7	4	7	5	3
3	1	3	5	2	1	1	3
4	(2)	4	1	-	3	7	6
5	(2)	(3)	4	6	-	2	4
6	2	4	3	5	-	6	3
7	(1)	5	3	6	3	7	-
8	7	4	5	(3)	6	2	5

$$\begin{aligned}
 & 5+3+2 + (1+3+3+3+2)/2 \\
 & + (2+2+2+1+3)/2 \\
 & = 21
 \end{aligned}$$

# ผลการใช้ Lower Bound แบบต่าง ๆ

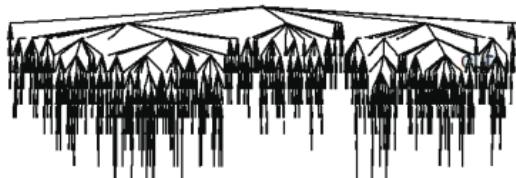
12 เมือง

$11! = 39,916,800$  ปมสถานะ



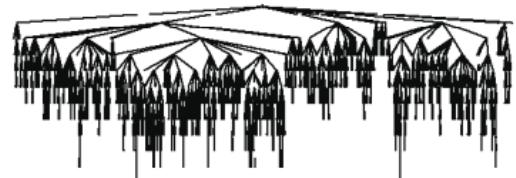
$\Sigma$  สั้นสุดพุงออก

3,760 ปม



$\Sigma$  สั้นสุดพุงเข้า

3,748 ปม



$\Sigma$  สั้นสุดพุงเข้าออก

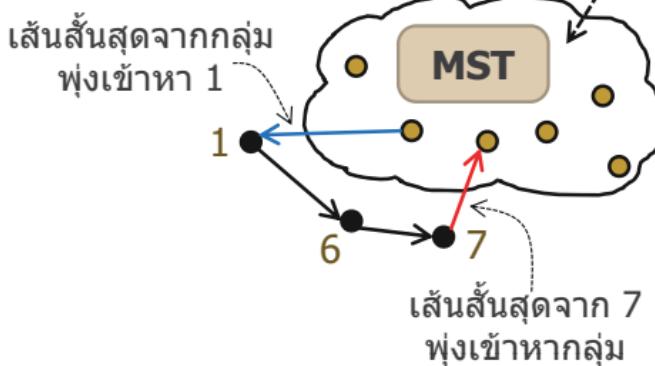
3,061 ปม

# ผลการใช้ Lower Bound แบบต่าง ๆ

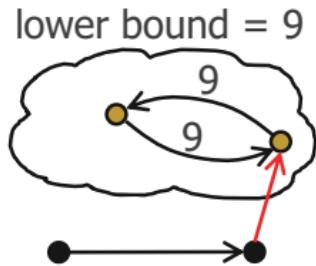
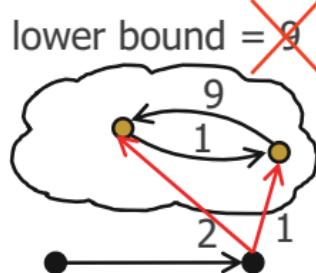
$n$	$(n-1)!$	จำนวนปมที่ค้นจนพบคำตอบที่ดีสุด		
		$\sum$ พุงออกสั้นสุด	$\sum$ พุงเข้าสั้นสุด	$\frac{1}{2} \sum$ พุงออกสั้นสุด + $\frac{1}{2} \sum$ พุงเข้าสั้นสุด
12	3.9E+07	3,760	3,748	3,061
13	4.8E+08	5,467	5,596	4,609
14	6.2E+09	13,829	14,391	11,438
15	8.7E+10	24,095	30,127	21,419
16	1.3E+12	212,427	53,906	76,597
17	2.1E+13	812,003	115,314	223,374
18	3.6E+14	817,545	126,305	227,961
19	6.4E+15	879,701	160,094	260,309
20	1.2E+17	25,302	3,203	6,547
21	2.4E+18	832,548	684,112	519,849
22	5.1E+19	1,743,490	759,293	709,383
23	1.1E+21	7,976,159	4,395,088	3,776,967
24	2.6E+22	8,792,218	6,736,808	4,622,674
25	6.2E+23	9,274,855	10,393,598	5,556,872
26	1.6E+25	16,497,668	15,663,247	8,426,109

# Lower Bound อีกแบบ : MST

ต้องการ lower bound ระยะทาง  
ของวิถีสั้นสุดที่ผ่านทุกปมในกลุ่ม  
(ขอ path ไม่ใช่ cycle)



ใช้กับกรณี  $w_{ij} = w_{ji}$   
(symmetric TSP)

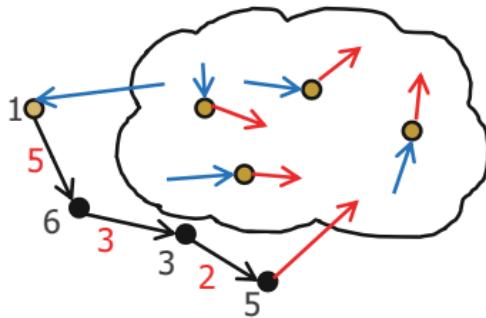


# เปรียบเทียบ Lower Bounds

$n$	$(n-1)!$	จำนวนปมที่ค้นจนพบคำตอบที่ดีสุด	
		$\frac{1}{2} \sum$ พุงออกสั้นสุด + $\frac{1}{2} \sum$ พุงเข้าสั้นสุด	ใช้ minimum spanning tree
12	3.9E+07	11,027	2,814
13	4.8E+08	15,605	4,739
14	6.2E+09	14,073	2,896
15	8.7E+10	47,407	8,780
16	1.3E+12	50,367	6,166
17	2.1E+13	3,536,828	530,603
18	3.6E+14	1,223,231	240,766
19	6.4E+15	1,653,389	239,125
20	1.2E+17	4,382,206	455,304

ใช้กับกรณี  $w_{ij} = w_{ji}$   
(symmetric TSP)

# ลองทำดู



เลือกตัวมากระหว่าง  
ผลรวมเส้นเชื่อมสั้นสุดที่ พุงเข้า  
ผลรวมเส้นเชื่อมสั้นสุดที่ พุงออก

# สรุป

## ❖ ลยทุกรูปแบบ

- ❖ DFS : ใช้หน่วยความจำน้อย แต่อาจหาคำตอบไม่พบ
- ❖ BFS : พบคำตอบที่ใกล้รากสุด แต่ใช้หน่วยความจำมาก
- ❖ LC-search : ใช้ cost function นำทางไปสู่คำตอบ

## ❖ Backtracking

- ❖ มีการพิจารณาว่าควรค้นต่อจาก state ปัจจุบันหรือไม่

## ❖ Branch and bound

- ❖ ใช้กับ optimization problem
- ❖ รักษาเลือกปมที่ดีเพื่อแทรกกิ่ง
- ❖ ใช้ขอบเขตของค่าของคำตอบที่ state ต่าง ๆ ในการเปรียบเทียบกับคำตอบดีสุดที่รู้มา เพื่อจำกัดการแทรกกิ่ง

# ເວັນພືບປົງ

ສມ່າຍ ປະສິທີງຕະກຸມ

# หัวข้อ

ปัญหาง่าย – ปัญหายาก

**Polynomial – Exponential**

การจัดกลุ่มปัญหา

กลุ่มปัญหา P และ NP

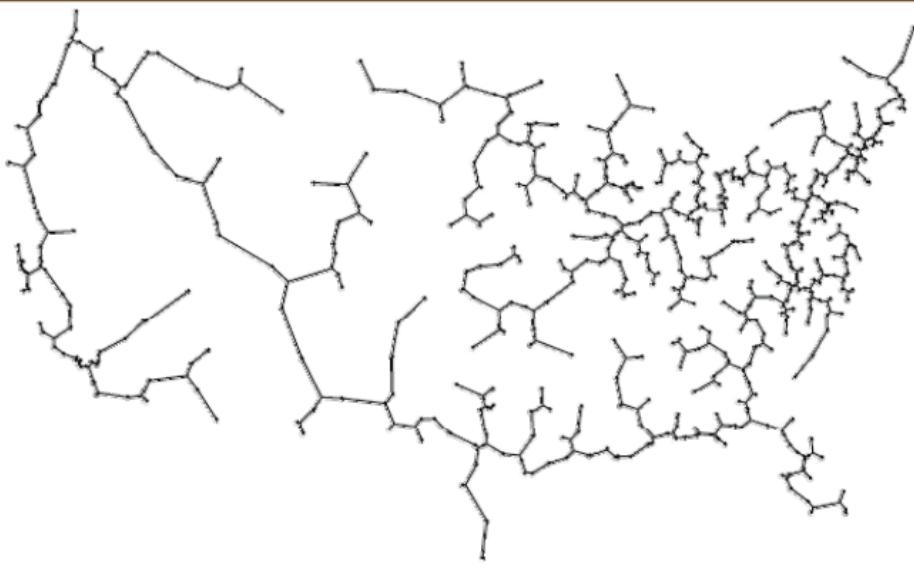
การลดรูปปัญหา

ปัญหา NP-complete

# เวลาในการหาคำตอบ

ปัญหา & วิธีแก้ไข	เวลาการทำงาน
ค้นข้อมูลในรายการที่เรียงแล้ว 1 ล้านตัว หาตัวน้อยสุด 1 ล้านตัว	< 1 ไมโครวินาที
เรียงลำดับข้อมูล 1 ล้านตัว	< 0.01 วินาที
หารากศักราชสั้นสุดในกราฟที่มีเส้นเชื่อมล้านเส้น	< 1 วินาที
Geosteiner หา Steiner tree ของ 532 ปม	2.5 ชม.
Selby & Riodan แก้ Eternity Puzzle (ประกอบแผ่นจำนวน 209 ชิ้น ให้เป็นรูป 12 เหลี่ยมด้านเท่า)	7 เดือน (2 CPUs)
Concorde แก้ TSP ที่มี 15,112 เมือง	เวลารวมของ CPU จำนวน 110 ตัว ≈ 20 ปี

# Steiner Tree : 532 ปม

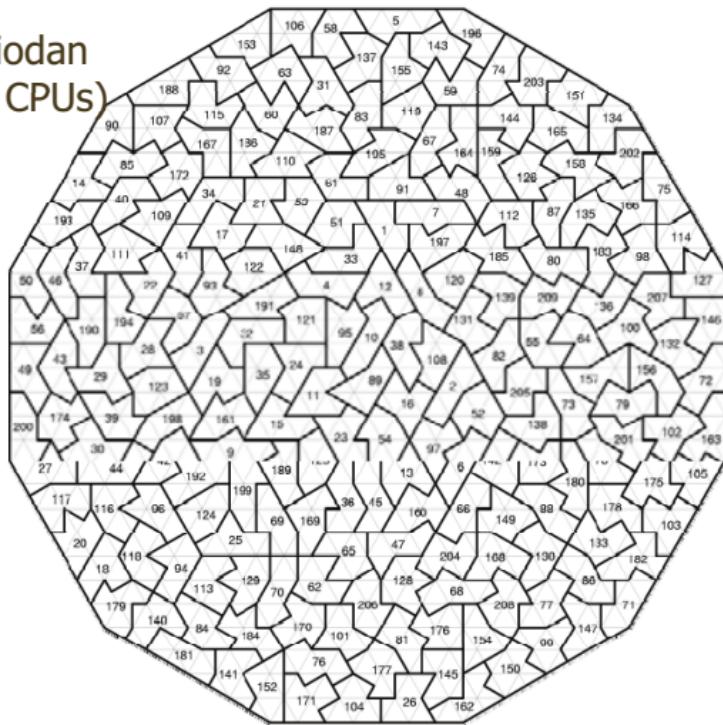


ใช้ GeoSteiner หาคำตอบของปัญหาขนาด 532 ปม  
ใช้เวลา 2.5 ชั่วโมง (ปี ค.ศ. 1998)

<http://www.diku.dk/hjemmesider/ansatte/martinz/geosteiner/>

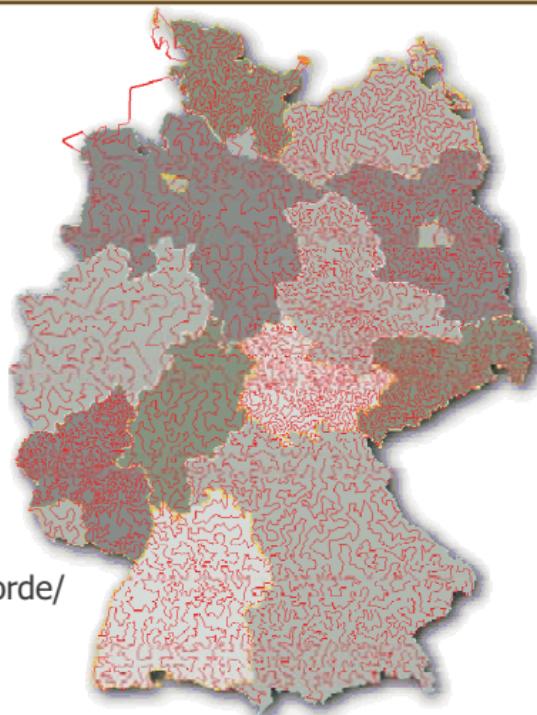
# Eternity Puzzle : 209 ชิ้น

Selby & Riodan  
7 เดือน (2 CPUs)



# TSP : 15,112 เมือง

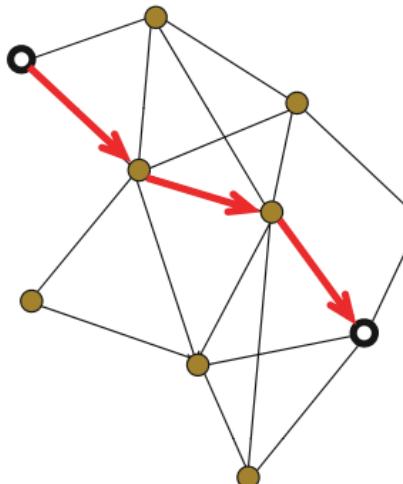
- ❖ หาคำตอบดีสุดได้ด้วย Concorde TSP
- ❖ ใช้ 110 CPUs
- ❖ รวมเวลา 585,936,700 s.  
(เทียบกับการใช้เครื่อง 500MHz ทำงาน 22.6 ปี)



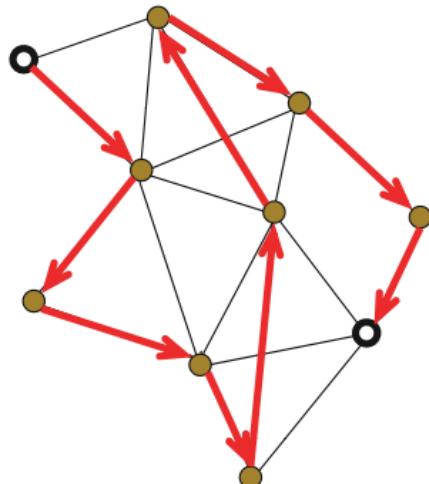
<http://www.tsp.gatech.edu/concorde/>

# ปัญหาง่าย – ยาก?

Shortest Path



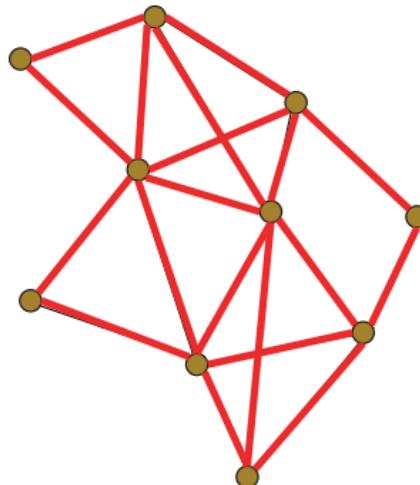
Longest Path



ทำไมถึงไม่หารวิถียาวสุด ด้วยการ  
ติดลบเส้นเชื่อม และหารวิถีสั้นสุด ?

# ปัญหาง่าย – ยาก?

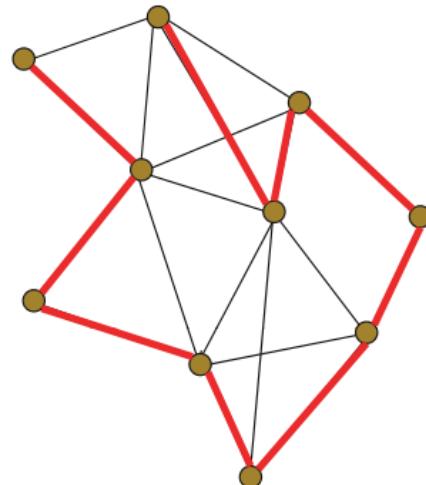
Euler Path



ผ่านทุกเส้น เส้นละครั้ง

ลำดับเส้น  $\rightarrow [n(n - 1)/2]$  !

Hamiltonian Path

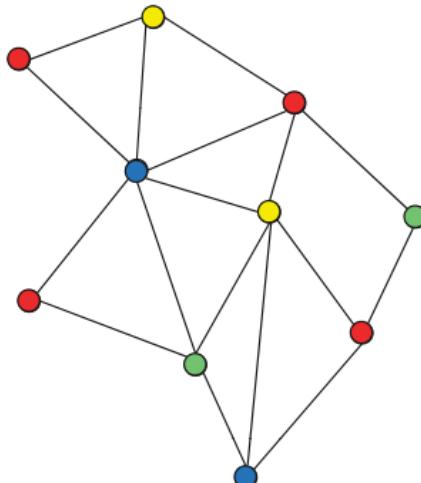


ผ่านทุกปม ปมละครั้ง

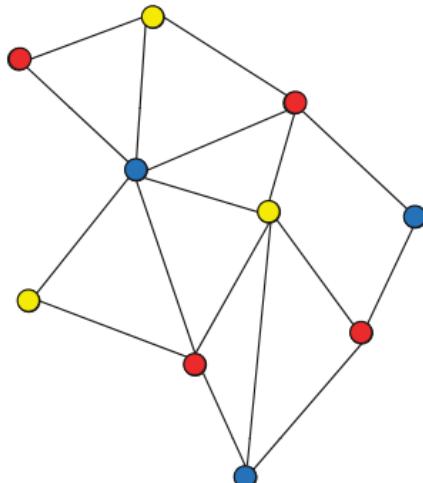
ลำดับปม  $\rightarrow n !$

# ปัญหาราย – ยาก?

Planar 4 color



Planar 3 color



Planar 2 color  
ก็ง่ายมาก ลองคิดดู

# ปัญหาง่าย – ยาก?

Primality Testing

Factoring

1111111111111111123

No

1111111111111111123

$$\begin{aligned} &= 8092579104960751 \\ &\quad \times 1373 \end{aligned}$$

# อัลกอริทึมที่มีและไม่มีประสิทธิภาพ

## ❖ Search

- ❖  $O(n)$  : sequential search → ช้า
- ❖  $O(\log n)$  : binary search → เร็ว

## ❖ Sorting

- ❖  $O(n^2)$  : bubble sort, insertion sort → ช้า
- ❖  $O(n \log n)$  : merge sort, heap sort → เร็ว

## ❖ Longest Common Subsequence

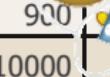
- ❖  $O(2^n)$  : brute force → ช้า
- ❖  $O(n^2)$  : dynamic programming → เร็ว

อะไรจะเป็นตัวบอกว่าอัลกอริทึมมีประสิทธิภาพ ?

# Polynomial vs. Exponential

- ❖ **Polynomial-time algorithms**      Efficient
  - ❖ อัลกอริทึมที่ใช้เวลาทำงานเป็น  $O(n^k)$   $k$  เป็นค่าคงตัว
- ❖ **Exponential-time algorithms**      Inefficient
  - ❖ อัลกอริทึมที่ใช้เวลาทำงานเป็น  $\Omega(c^n)$   $c$  เป็นค่าคงตัว  $c > 1$

ฟังก์ชัน	n				
	10	20	30	100	200
$2n$	20	40	60	200	400
$n \log_2 n$	33	86	664	1528	
$n^2$	100	400	900	10000	40000
$n^4$	10000	40000	810000	$1 \times 10^8$	$1.6 \times 10^9$
$2^n$	1024	1048576	$1.07 \times 10^{30}$	$1.27 \times 10^{30}$	$1.61 \times 10^{60}$
$n!$	3628800	$2.43 \times 10^{18}$	$2.65 \times 10^{157}$	$3.3 \times 10^{157}$	$7.89 \times 10^{374}$
$n^k$	$10^{10}$	$1.05 \times 10^{26}$	$2.06 \times 10^{74}$	$1 \times 10^{200}$	$1.61 \times 10^{460}$



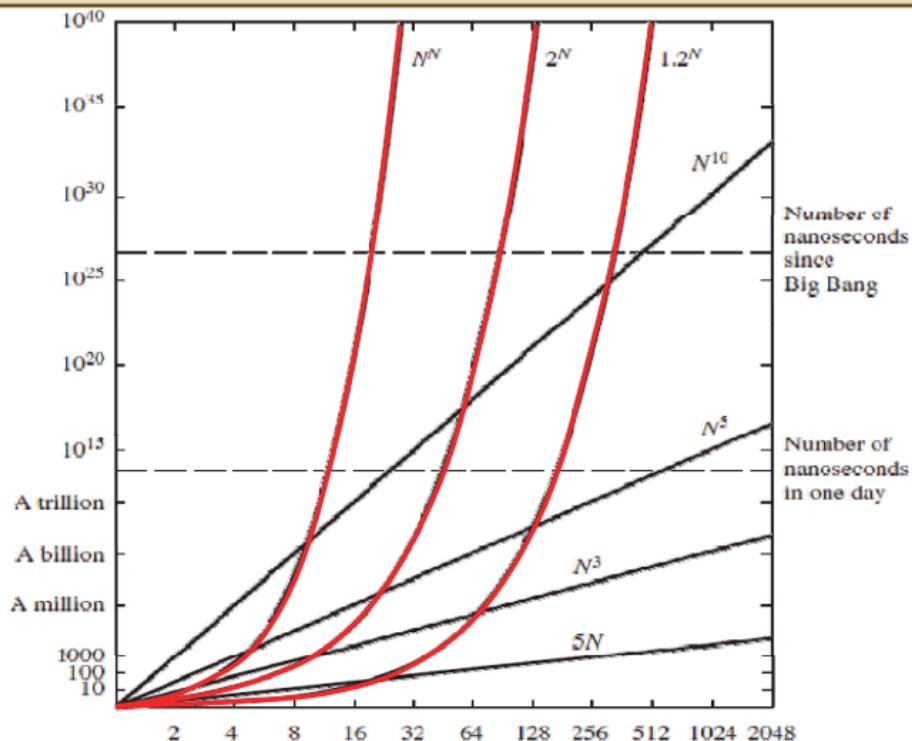
# เมื่อ $n$ ใหญ่ : Exp-time Algo ซ้ามาก

ให้ CPU ทำงาน หมื่นล้านคำสั่งใน 1 วินาที

$$10 \text{ Giga} = 10^{10}$$

ฟังก์ชัน	$n$				
	10	20	30	100	200
$2n$	2ns	4ns	6ns	20ns	40ns
$n \log_2 n$	3.3ns	8.6ns	15ns	15ns	153ns
$n^2$	10ns	40ns	90ns	1 $\mu$ s	4 $\mu$ s
$n^4$	1 $\mu$ s	4 $\mu$ s	81 $\mu$ s	0.01s	0.16s
$2^n$	102ns	10.5ms	0.11s	$4.0 \times 10^{12}$ y	$5.1 \times 10^{42}$ y
$n!$	36.3ms	7.7 y	$8.4 \times 10^{14}$ y	$2.9 \times 10^{140}$ y	$2.5 \times 10^{357}$ y
$n^n$	1s	$330 \times 10^6$ y	$6.5 \times 10^{26}$ y	$3.2 \times 10^{182}$ y	$5.1 \times 10^{388}$ y

# Polynomial vs. Exponential



# อย่ากังวลมากกับกรณีเปลก ๆ

- ❖ ถ้า  $k$  มาก  $O(n^k) - \text{algo. ก็ช้ามากเหมือนกัน ?}$
- ❖  $n^{10}$  กับ  $n^{\log \log n}$  น่าใช้ อันไหน ?
  - ❖  $n^{10}$  : polynomial
  - ❖  $n^{\log \log n}$  : super-polynomial
  - ❖ ถ้า  $n = 10^{100}$  :  $n^{10} = 10^{1000}$ ,  $n^{\log \log n} = 10^{200}$
- ❖ เพื่อความง่ายในการจัดกลุ่มปัญหา
  - ❖ Poly-time algo. (ส่วนใหญ่)  $\rightarrow$  efficient algo.
  - ❖ Exp-time algo. (ส่วนใหญ่)  $\rightarrow$  inefficient algo.

# ปัญหาง่าย - ยาก

## ❖ Tractable problems

- มี polynomial-time algo.
- หากำลังดี
  - min, max,
  - sort, search, select,
  - shortest path, MST, LCS,
  - ...

ปัญหาง่าย

## ❖ Intractable problems

- ต้องใช้อย่างน้อย

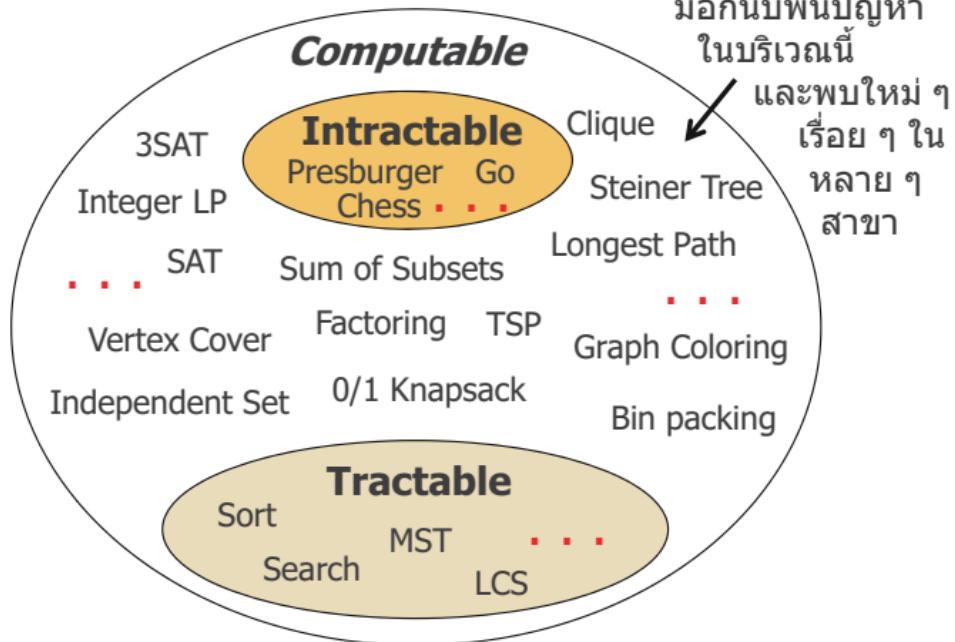
### Super-Polynomial algo.

- Tower of hanoi,
- Halting in k instructions
- Generalized Chess, Checker

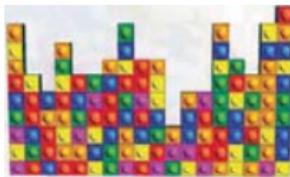
ปัญหายาก

แล้วพวก TSP, 0/1 Knapsack ?

# การจัดกลุ่มปัญหา



# Games & Puzzles : ຢາກ?



Clickomania

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

Sudoku



Rush Hour



Minesweeper



Tiling (Edge-Matching)

SEND      9567  
+MORE      +1085  
MONEY      10652

Cryptarithms

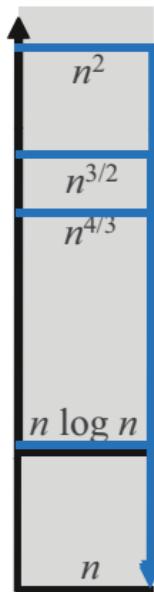
# Algorithmic Gap

## Comparison-based Sorting

พิสูจน์ lower bound  
ของเวลาการทำงานใน  
การแก้ปัญหาที่สูง ๆ

ต้นไม้จำลองการ sort  
มี  $n!$  ใบ จึงสูง  $\geq \log_2 n!$   
ต้องเปรียบเทียบ  $\Omega(n \log n)$

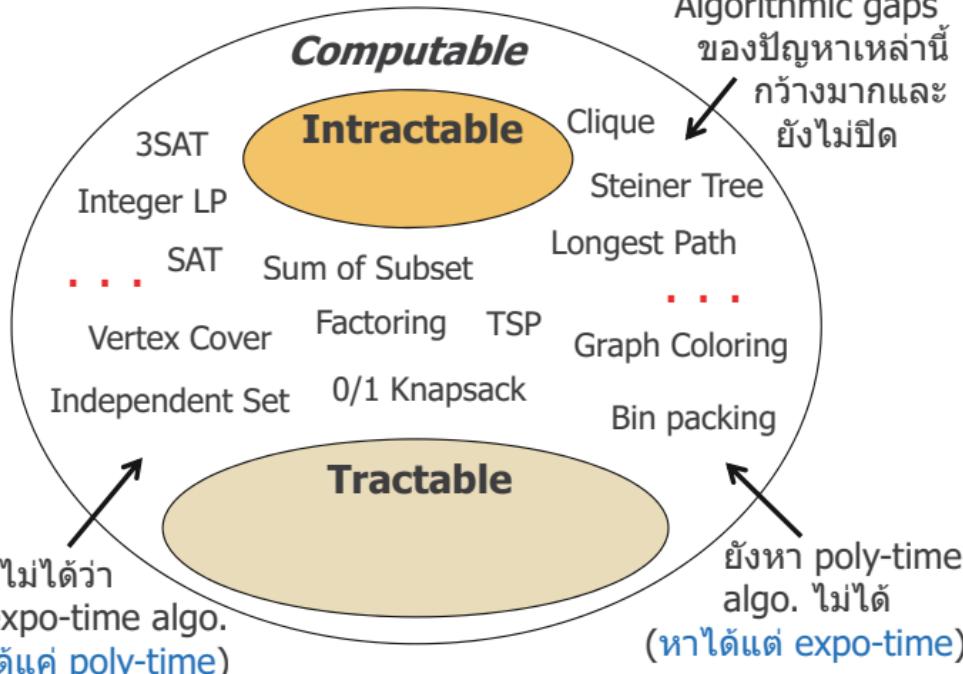
sort ต้องพิจารณา  
ข้อมูลทุกดัว  $\Omega(n)$



- $\Theta(n^2)$  : selection sort
- $O(n^2)$  : insertion sort
- $O(n^2)$  : Shell sort (Shell)
- $O(n^{3/2})$  : Shell sort (Knuth)
- $O(n^{4/3})$  : Shell sort (Sedgewick)
- $O(n \log n)$  : Merge sort

อุกแบบอัลกอริทึมที่มี  
upper bound ของ  
เวลาการทำงานที่ต่ำ ๆ

# Algorithmic Gap



# Decision Problems

- ❖ ปัญหาตัดสินใจ ที่ให้ผลลัพธ์เป็นหนึ่งในสองค่า
  - ❖ True / False, Yes / No, มี / ไม่มี, ได้ / ไม่ได้
- ❖ ตัวอย่าง
  - ❖ G มี spanning tree ที่เส้นเชื่อมยาวรวมไม่เกิน k หรือไม่
  - ❖ แบ่งเซต D เป็นสองเซตย่อยที่มีผลรวมข้อมูลเท่ากันได้หรือไม่
  - ❖ P เป็นจำนวนเฉพาะหรือไม่
  - ❖ ...
- ❖ จากนี้ไปจะสนใจปัญหาตัดสินใจ
  - ❖ สนใจแต่ขนาดของ input ไม่ต้องสนใจขนาดของ output
  - ❖ ง่ายต่อการเปรียบเทียบความยากง่ายของปัญหา
  - ❖ ง่ายในการจัดกลุ่มปัญหา

# Decision Problems

## ❖ ปัญหาที่ผ่านมา ให้ผลลัพธ์ที่ไม่ใช้แบบ True/False

- ❖ แต่สามารถแปลงเป็น decision problem ได้

## ❖ Sum of subsets

- ❖ หาเซตย่อยของ  $D$  ที่ให้ผลรวมเป็น  $k$
- ❖  $D$  มีเซตย่อยที่มีผลรวม เป็น  $k$  หรือไม่

## ❖ Travelling Salesman

- ❖ หา tour ที่มีระยะทางสั้นสุด ที่ผ่านทุกเมืองในกราฟ  $G$
- ❖ กราฟ  $G$  มี tour ผ่านทุกเมือง ที่มีระยะทางรวม อย่างมาก  $k$  หรือไม่

## ❖ 0/1 Knapsack

- ❖ หาของที่หยิบใส่ถุง (ไม่ขาด) ให้มีมูลค่ารวมสูงสุด
- ❖ มีวิธีหยิบของใส่ถุง (ไม่ขาด) ได้มูลค่ารวม อย่างน้อย  $k$  หรือไม่

optimization  
problems

## Decision กับ Optimization

- ❖ ให้  $Q_{DEC}$  เป็นปัญหาตัดสินใจของ  $Q_{OPT}$
- ❖ ถ้า  $Q_{DEC}$  ยาก  $\rightarrow Q_{OPT}$  ยากแน่
- ❖ ถ้า  $Q_{OPT}$  ง่าย  $\rightarrow Q_{DEC}$  ง่ายด้วย
- ❖ มีหลายปัญหาที่
  - ❖ ใช้ algo ของ  $Q_{DEC}$  หากคำตอบของ  $Q_{OPT}$  ได้

# Graph Coloring

❖ **Graph Coloring** ปั่นปลายของเส้นเชื่อมเดียวกัน ห้ามมีสีเหมือนกัน

- ❖  $GC_{OPT}$  : ใช้กีสีน้อยสุดในการทาปั่นของกราฟ  $G$  ( หา  $\chi(G)$  )
- ❖  $GC_{DEC}$  : ใช้  $k$  สีทาปั่นของกราฟ  $G$  ได้หรือไม่

```
GC_Dec( G, k ) {  
    c = GC_Opt( G )  
    return c <= k  
}
```

ใช้ algo ของ  $GC_{OPT}$   
หากคำตอบให้  $GC_{DEC}$  ได้

```
GC_OPT( G=(V,E) ) {  
    for (k = 1; k<=|V|; k++) {  
        if (GC_Dec(G, k)) return k  
    }  
    return ???  
}
```

ใช้ algo ของ  $GC_{DEC}$   
หากคำตอบให้  $GC_{OPT}$  ได้

# หาค่าต่ำสุดของ $GC_{Opt}$ ด้วย $GC_{Dec}$

```
GC_OPT( G=(V,E) ) {
    low = 1; high = |V|; k = high
    while (low <= high) {
        mid = (low + high) / 2
        if (GC_Dec(G, mid)) {
            if (mid < k) k = mid
            high = mid - 1
        } else {
            low = mid + 1
        }
    }
    return k
}
```

ต.ย. ให้กราฟมี 50 ปุ่ม  
 $chromatic\ number = 5$

low	high	mid	$GC_{Dec}$	k
1	50	25	true	25
1	24	12	true	12
1	11	6	true	6
1	5	3	false	6
4	5	4	false	6
5	5	5	true	5

## ลองคิดดู

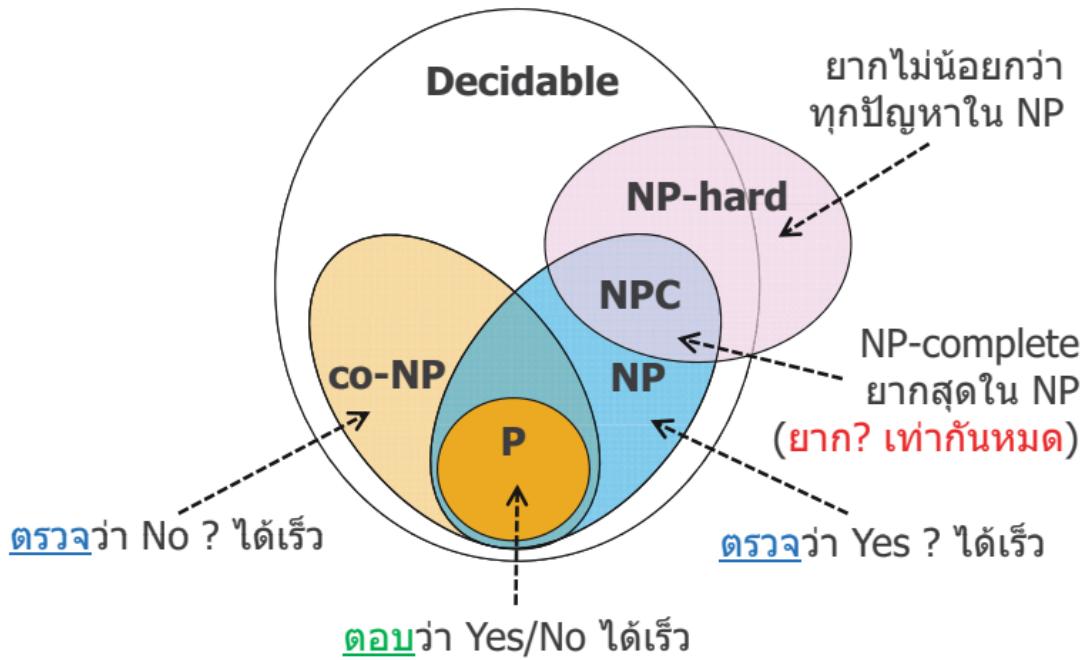
### ❖ กำหนดให้มี **DSubsetSum( d[1..n], k )**

- ❖ คืน true : เมื่อมีเซตย่อยของ  $d$  ที่มีผลรวมเท่ากับ  $k$
- ❖ คืน false : เมื่อไม่มีเซตย่อยของ  $d$  ที่มีผลรวมเป็น  $k$

### ❖ จงเขียน **SubsetSum( d[1..n], k )**

- ❖ คืนเซตย่อยของ  $d$  ที่มีผลรวมเป็น  $k$  (ถ้ามี)
- ❖ ใช้ DSubsetSum(  $d, k$  ) ให้เป็นประโยชน์
- ❖ ถ้า DSubsetSum ทำงานใน polynomial time  
ต้องการให้ SubsetSum ทำงานใน polynomial time ด้วย

# การจัดกลุ่มปัญหา



"is NP-hard" - Google Scholar - Mozilla Firefox

File Edit View History Bookmarks Tools Help

http://scholar.google.co.th/scholar?q="is+NP-hard"&hl=th&client=firefox

"is NP-hard"

"is NP-hard" - Google Scholar

ค้นหา

สืរคำ Pig Latin  
ภาษาหนอนค่า Scholar

วินาที

# Google scholar

"is NP-hard"

Protein design is NP-hard  
Optimal fractal coding is NP-hard  
Clique-width minimization is NP-hard  
Recognizing tough graphs is NP-hard  
Unit disk graph recognition is NP-hard  
Learning Bayesian networks is NP-hard  
Minimum-weight triangulation is NP-hard  
Checking robust nonsingularity is NP-hard  
Computing variance for interval data is NP-hard  
Precise flow-insensitive may-alias analysis is NP-hard  
...

Learning Bayesian networks is NP-hard

DM Chickering, D Geiger, D Heckerman - Microsoft Research, 1994 - Citeseer

## การอุปกรณ์แบบอัลกอริทึม

"is NP-Complete" - Google Scholar - Mozilla Firefox

File Edit View History Bookmarks Tools Help

http://scholar.google.co.th/scholar?start=0&q="is+NP-Complete"&hl=th! "is NP-hard"

"is NP-Complete" - Google Scholar

Google scholar "is NP-Complete" ค้นหา สืบค่า Pig Latin การค้นหาใน Google Scholar 9 วินาที

Scholar

คลิกลับ : ค้นหา

[PDF] Comp... M Yannakakis ... 1981 So COMPUTIN show that th อ้างโดย 493

[PDF] Cons... L Hyafil, RL Volume 5, n Theorem. D machine ca อ้างโดย 369

Tensor rank is NP-complete  
Wire-routing is NP-complete  
Minesweeper is NP-complete  
Cyclic ordering is NP-complete  
Crossing number is NP-complete  
Generalized Hi-Q is NP-complete  
Weapons allocation is NP-complete.  
Elastic image matching is NP-complete  
Positive first order logic is np-complete  
Min cut is NP-complete for edge weighted trees  
Topology of strings: Median string is NP-complete  
...

# กลุ่ม P

- ❖ P คือ กลุ่มของปัญหาตัดสินใจที่ หาคำตอบได้ ในเวลา polynomial time
- ❖ ตัวอย่าง
  - ❖ รายการ L มีตัวหนึ่งมากหรือไม่ (majority) ?
  - ❖ รายการ L มีข้อมูลที่มีค่าเป็น x หรือไม่ ?
  - ❖ กราฟ G มี spanning tree ที่ความยาวเส้นเชื่อมรวมไม่เกิน k ?
  - ❖ กราฟ G มีวิถี Euler หรือไม่ ?
  - ❖ ลำดับ X และ Y มีความยาวของ  $LCS(X, Y)$  อย่างน้อย k ?
  - ❖ การคูณเมตริกซ์  $A_1A_2\dots A_n$  ใช้จำนวนการคูณแบบสเกลาร์ไม่เกิน k ครั้งหรือไม่ ?
  - ❖ M เป็นจำนวนเฉพาะหรือไม่ ?
  - ❖ M เป็นจำนวนประกอบหรือไม่ ?

# กลุ่ม NP

- ❖ P : กลุ่มของปัญหาตัดสินใจที่ หาคำตอบได้ ในเวลา polynomial time
- ❖ NP : กลุ่มของปัญหาตัดสินใจที่ ตรวจคำตอบ yes ได้ ในเวลา polynomial time

d มีตัวหามากหรือไม่ ?

```
hasMajority(d[1..n]) {
    for (i=1; i<=n; i++) {
        c = count(d, d[i])
        if (c > n/2) return true
    }
    return false
}
```

$O(n^2)$

MAJORITY  $\in P$

# กลุ่ม NP

- ❖ P : กลุ่มของปัญหาตัดสินใจที่ หาคำตอบได้ ในเวลา polynomial time
- ❖ NP : กลุ่มของปัญหาตัดสินใจที่ ตรวจคำตอบ yes ได้ ในเวลา polynomial time

d มีตัวหามากหรือไม่ ?

```
validMajority(d[1..n], maj) {  
    c = 0;  
    for (i=1; i<=n; i++)  
        if (d[i] == maj) c++  
    return c > n/2  
}
```

witness,  
certificate

$\Theta(n)$

MAJORITY ∈ NP

$P \subseteq NP$

(polynomial time verifiable)

# Verifier : Graph Coloring

เราทาสีปมใน G ด้วยสี k สีได้หรือไม่ ?

witness,  
certificate

```
validColoring(G[1..n][1..n], k, color[1..n]) {
    numberOfColors = max(color)
    if (numberOfColors > k) return false
    for (i=1; i<n; i++) {
        for (j=i+1; j<=n; j++) {
            if (G[i][j] == true AND color[i] == color[j])
                return false
        }
    }
    return true
}
```

$O(n^2)$

GRAPHCOLORING  $\in$  NP

# Verifier : Sum of Subset

d มีเซตย่อยที่มีผลรวมเท่ากับ k ?

witness,  
certificate

```
validSubsetSum( d[1..n], k, x[1..n] ) {  
    sum = 0  
    for (i=1; i<=n; i++)  
        sum += d[i]*x[i]  
    return (sum == k)  
}
```

$\Theta(n)$

SUBSETSUM  $\in$  NP

# Verifier : TSP

$G$  มี Hamiltonian circuit ที่ยาว  $\leq k$  ? witness, certificate

```

validTSP( G[1..n][1..n], k, v[1..m] ) {
    if ( m != n ) return false
    visited = an array of boolean[1..n]
    visited[] = false; visited[v[1]] = true
    for (i=2; i<=n; i++) {
        if ( visited[v[i]] ) return false
        visited[v[i]] = true
        length += G[v[i-1]][v[i]]
    }
    length += G[v[n]][v[1]]
    return length <= k
}

```

 $\Theta(n)$ 

TSP  $\in$  NP

# ให้แสดงว่า KNAPSACK ∈ NP

มีวิธีหยັບຂອງໃສຄຸງເປົ້າ ໂດຍທີ່ຄຸງໄມ່ຂາດ  
ແລະໄດ້ມຸລรวมรวมອ່າງນົບຍ k ?

```
validKnapsack(    ???    ) {  
    ???  
}
```

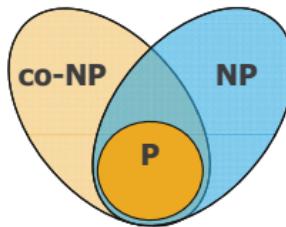
# กลุ่ม co-NP

## ❖ กลุ่มปัญหา NP :

- ❖ กลุ่มของปัญหาตัดสินใจที่ ตรวจคำตอบ yes ได้  
ในเวลา polynomial time

## ❖ กลุ่มปัญหา co-NP :

- ❖ กลุ่มของปัญหาตัดสินใจที่ ตรวจคำตอบ No ได้  
ในเวลา polynomial time



# Primality Testing

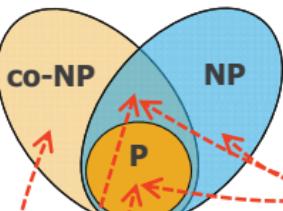
ตรวจ Yes : n เป็นจำนวนเฉพาะ ?

Pratt certificate

```
validPrime( n, c ) {  
    // คืน true ถ้าเป็น prime  
    // ขอไม่แสดงรายละเอียด  
}
```

Prime  $\in$  NP

$\Theta( (\log n)^4 )$



ตรวจ No : n ไม่ใช่เป็นจำนวนเฉพาะ ? ตัวประกอบของ n

```
notValidPrime( n, c ) {  
    // คืน true ถ้าไม่ใช่ prime  
    return n % c == 0  
}
```

Prime  $\in$  co-NP

$\Theta( (\log n)^2 )$

ปี ค.ศ. 2006 Agrawal–Kayal–Saxena ได้พิสูจน์ว่า Prime  $\in$  P

# 3-Colorability

ตรวจ Yes : เราทาสีปมใน G ด้วย 3 สีได้ ใช่หรือไม่ ?

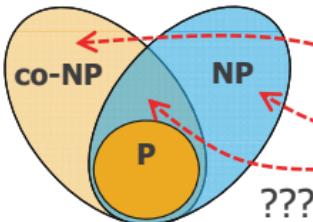
```
valid3Coloring( G[1..n][1..n], color[1..n] ) {
    ☺ ☺ // คืน true เมื่อทา 3 สีได้
}
```

3-Color  $\in$  NP

ตรวจ No : เราทาสีปมใน G ด้วยสี 3 สีไม่ได้ ใช่หรือไม่ ?

```
notValid3Coloring( G[1..n][1..n], ??? ) {
    ☹ ☹ // คืน true เมื่อทา 3 สีไม่ได้
}
```

3-Color  $\notin$  co-NP



ตรวจคำตอบ Yes	ตรวจคำตอบ No
ยาก	ง่าย
ง่าย	ยาก
ง่าย	ง่าย
ยาก	ยาก

# NP : Nondeterministic Poly...

- ❖ NP : กลุ่มของปัญหาตัดสินใจที่ ตรวจคำตอบ yes ได้ ในเวลา polynomial time
- ❖ NP : กลุ่มของปัญหาตัดสินใจที่ หาคำตอบ ได้ ด้วย Nondeterministic Polynomial-time algorithm

```
ND_TSP( G[1..n][1..n], k ) {  
    v = a new array[1..n]           เดาคำตอบ  
    for (i=1; i<=n; i++) v[i] = ND_Choice(1, n)  
    if ( validTSP(G, k, v) ) return true  
    FAIL  
}
```

ตรวจคำตอบ

# ND\_SubsetSum

```
ND_SubsetSum( d[1..n], k ) {  
    x = a new array[1..n]  
    for (i=1; i<=n; i++) {  
        x[i] = ND_Choice(0, 1)  
    }  
    if ( validSubsetSum(d, k, x) ) return true  
    FAIL  
}
```

เดาค่าตอบ

ตรวจค่าตอบ

# ND\_Coloring

```
ND_Coloring( G[1..n][1..n], k ) {  
    c = a new array[1..n]  
    for (i=1; i<=n; i++) {  
        c[i] = ND_Choice(1, k)  
    }  
    if ( validColoring(g, k, c) ) return true  
    FAIL  
}
```

เดาค่าตอบ

ตรวจค่าตอบ

# ลองเขียนเอง : ND\_Knapsack

```
ND_Knapsack( w[1..n], v[1..n], W, k ) {
```

เดาค่าตอบ

```
}
```

ตรวจค่าตอบ

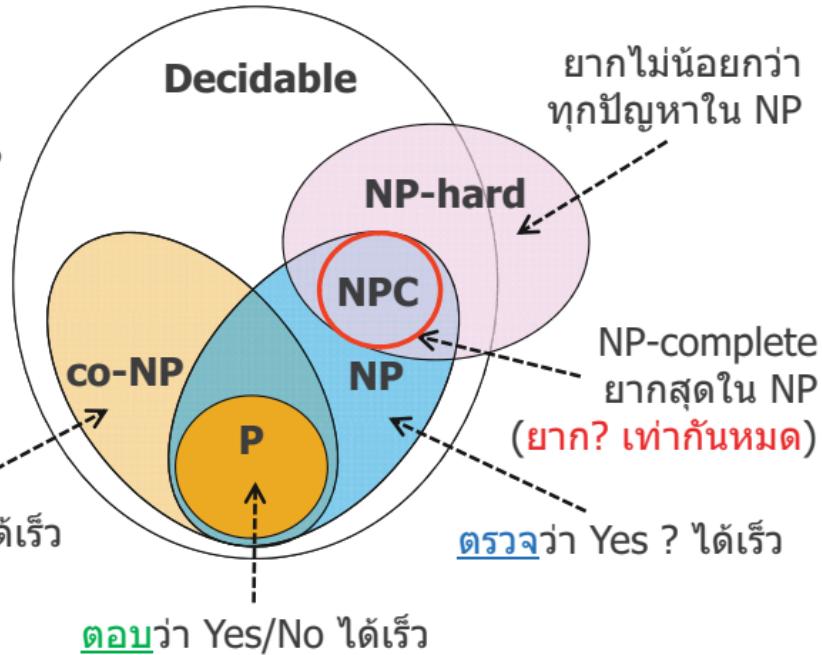
# NP-Complete

$P \subseteq NP$   
 $P \subseteq co-NP$   
 $P = NP ?$   
 $P = co-NP ?$   
 $NP = co-NP ?$

เชื่อกันว่า

$P \neq NP$   
 $P \neq co-NP$   
 $NP \neq co-NP$

ตรวจว่า No ? ได้เร็ว



# Reduction : การลดรูป



reduce  $q_1$  to  $q_2$

$$q_1 \leq_p q_2$$

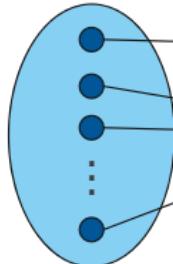
$q_1$  ไม่ยากกว่า  $q_2$

การแปลงใช้เวลาเป็น polynomial

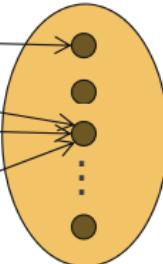
(polynomially reducible)

แปลงทุก instance ของ  $q_1$  ไปเป็น instance ของ  $q_2$

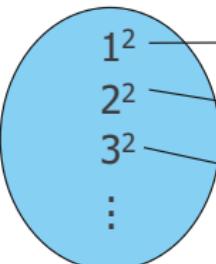
ปัญหา  $q_1$



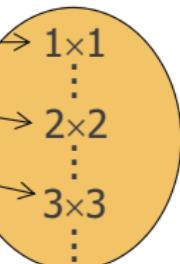
ปัญหา  $q_2$



ยกกำลังสอง



การคูณ



# Reduction : SQR กับ MULT

```
Sqr( x ) {  
    a = x, b = x  
    return Mult( a, b )  
}
```

SQR  $\leq_p$  MULT

```
Mult( a, b ) {  
    x = a+b, y = a-b  
    return (Sqr(x)-Sqr(y)) / 4  
}
```

MULT  $\leq_p$  SQR

$$a^2 + 2ab + b^2 = (a+b)^2$$

$$a^2 - 2ab + b^2 = (a-b)^2$$

$$4ab = (a+b)^2 - (a-b)^2$$

$$ab = \frac{(a+b)^2 - (a-b)^2}{4}$$

# Reduction : เพื่อใช้แก้ปัญหาใหม่

ปัญหา A ลดรูปเป็นปัญหา B ได้  
ใช้อัลกอริทึมของ B หาคำตอบให้ A ได้

instance ของปัญหา A



การแปลง instance ของ A  
เป็น instance ของ B

instance ของปัญหา B



อัลกอริทึมที่แก้ปัญหา B

คำตอบ ของปัญหา B



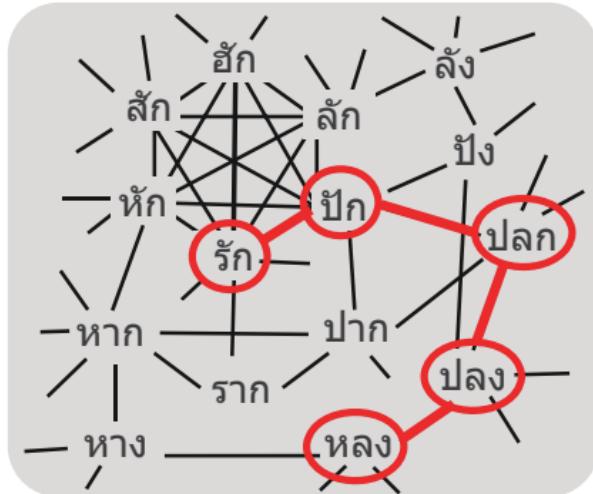
การแปลงคำตอบของ B  
เป็นคำตอบของ A

คำตอบของปัญหา A



# Word Ladder $\leq_p$ Path Finding

รัก → ปัก → ปลอก → ปลง → หลง



พจนานุกรม, คำตั้น, คำปลาย

สร้างกราฟจากคำศัพท์ใน  
พจนานุกรม

กราฟ + ปมต้น + ปมปลาย

Breadth First Search

วิธีจากปมต้น ไปปมปลาย

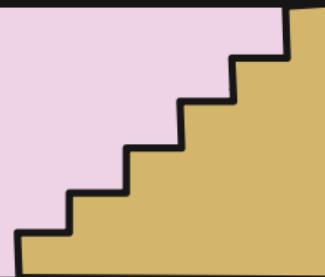
แปลงปมในวิธีเป็นลำดับ  
ของคำในแต่ละคำ

บันไดคำ

# Reduction : เพื่อใช้แก้ปัญหาใหม่

B

อยากรดีมชา



# Reduction : เพื่อใช้แก๊ส楠หาใหม่



# Reduction : เพื่อเทียบความยากง่าย

ปัญหา A ลดรูปเป็นปัญหา B ได้

ถ้า B มี poly-time algo.  
A มี poly-time algo. ด้วย

ถ้า B เป็นปัญหาง่าย  
A ก็ต้องง่ายด้วย

ถ้า A เป็นปัญหายาก  
B ก็ต้องยากด้วย

$A \leq_p B$   
A ไม่ยากกว่า B

# ความยากง่ายของ SQR กับ MULT

```
Sqr( x ) {  
    a = x, b = x  
    return Mult( a, b )  
}
```

SQR  $\leq_p$  MULT

SQR ไม่ยากกว่า MULT

```
Mult( a, b ) {  
    x = a+b, y = a-b  
    return (Sqr(x)-Sqr(y)) / 4  
}
```

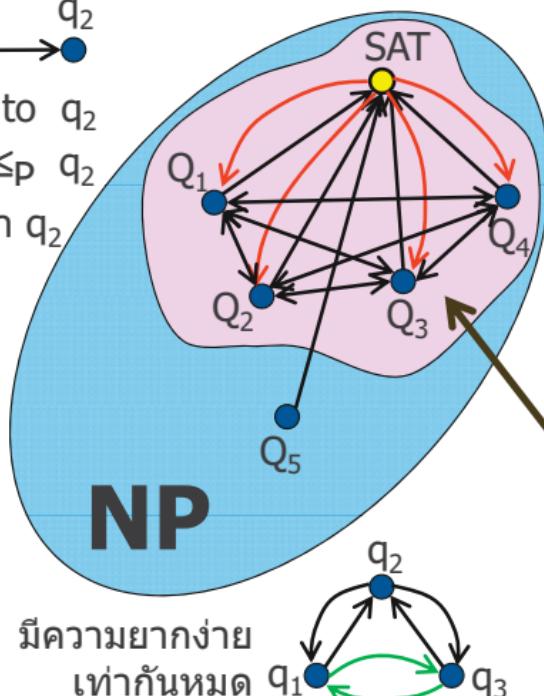
MULT  $\leq_p$  SQR

MULT ไม่ยากกว่า SQR

MULT กับ SQR  
ยาก (ง่าย) เท่ากัน

# NP-Complete

$q_1 \rightarrow q_2$   
 reduce  $q_1$  to  $q_2$   
 $q_1 \leq_P q_2$   
 $q_1$  ไม่ยากกว่า  $q_2$



Cook-Levin Theorem

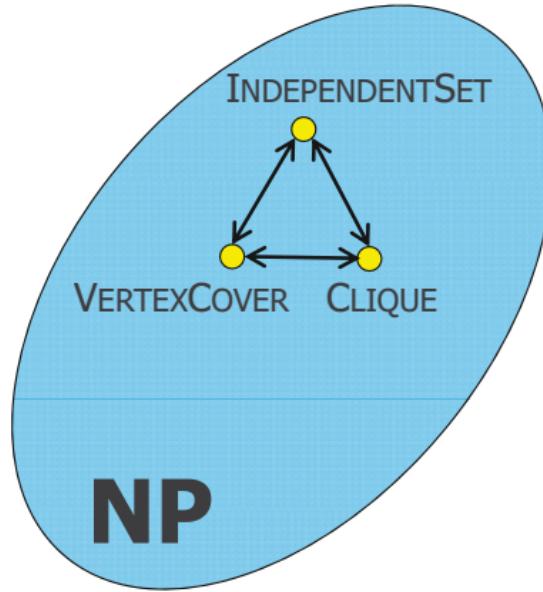
ทุกปัญหาใน NP  
 สามารถลดรูปไปเป็น  
 ปัญหา SAT ได้หมด

$\forall q \in NP, q \leq_P SAT$

SAT เป็นปัญหา  
 ยากที่สุดใน NP

**NP-Complete :**  
 กลุ่มปัญหาใน NP ที่  
 ยากที่สุดใน NP  
 (ยาก? เท่ากันหมด)

# Reduction : ตัวอย่าง



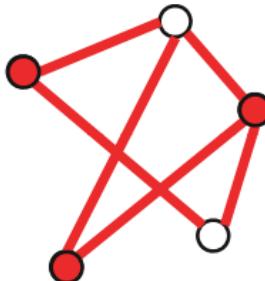
# Vertex Cover

## ❖ Optimization : input = G

- ❖ หาเซตย่อยของปม ( $V_C$ ) ในกราฟ G ที่มีขนาดเล็กสุด  
ที่เส้นเชื่อมทุกเส้นใน G มีจุดปลาย (อย่างน้อย 1 จุด) ใน  $V_C$

## ❖ Decision : input = G, k

- ❖ มีเซตย่อยของปม ( $V_C$ ) ในกราฟ G ขนาดอย่างมาก k หรือไม่  
ที่เส้นเชื่อมทุกเส้นใน G มีจุดปลาย (อย่างน้อย 1 จุด) ใน  $V_C$



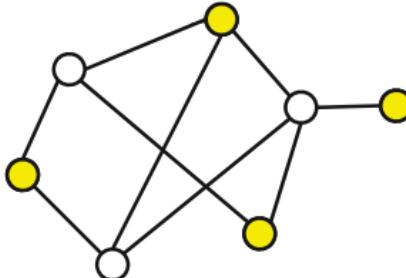
# Independent Set

## ❖ Optimization : input = G

- ❖ หาเซตย่อยของปม ( $V_I$ ) ในกราฟ G ที่มีขนาดใหญ่สุด ที่ไม่มีเส้นเชื่อมใน G ต่อระหว่างคู่ปมใด ๆ ใน  $V_I$

## ❖ Decision : input = G, k

- ❖ มีเซตย่อยของปม ( $V_I$ ) ในกราฟ G ขนาดอย่างน้อย k หรือไม่ ที่ไม่มีเส้นเชื่อมใน G ต่อระหว่างคู่ปมใด ๆ ใน  $V_I$



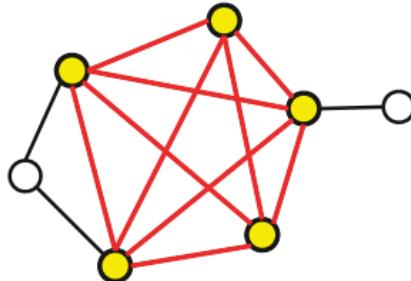
# Clique

## ❖ Optimization : input = G

- ❖ หาก Graf ย่ออยู่ในกราฟ (complete subgraph) ที่มีขนาดใหญ่สุดในกราฟ G

## ❖ Decision : input = G, k

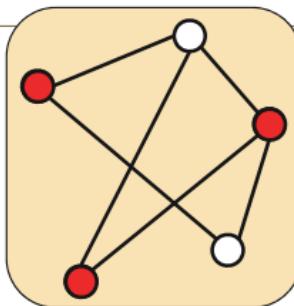
- ❖ G มีกราฟย่ออยู่ในกราฟขนาดอย่างน้อย k ปุ่ม ?



# VERTEXCOVER ∈ NP

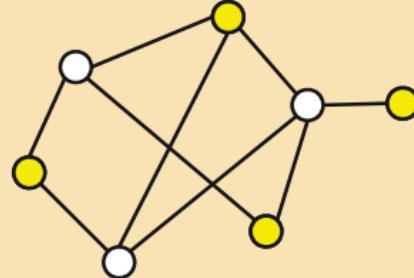
```
validVertexCover( G=(V,E), k, vc[1..n] ) {  
    if (n > k) return false  
    covered[] = false  
    for (i=1; i<=n; i++)  
        for each u ∈ Adj( vc[i] )  
            covered[ (vc[i],u) ] = true  
    for each edge (u,v) ∈ E  
        if ( NOT covered[ (u,v) ] ) return false  
    return true  
}
```

$O(v^2)$



# INDEPENDENTSET $\in$ NP

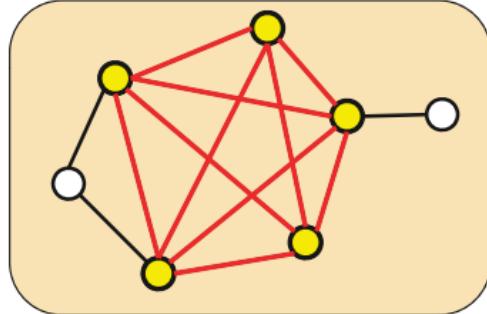
```
validIndependentSet( G=(V,E) , k, vi[1..n] ) {  
    if (n < k) return false  
    for (i=1; i<=n; i++)  
        for (j=i+1; j<=n; i++)  
            if ( (vi[i], vi[j])  $\in$  E ) return false  
    return true  
}
```

 $O(v^2)$ 

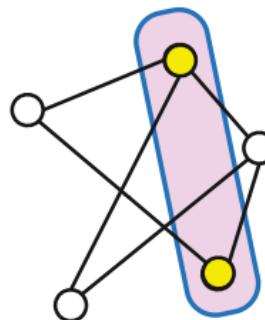
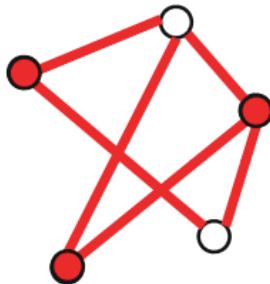
# CLIQUE $\in$ NP

```
validClique( G=(V,E) , k, c[1..n] ) {  
    if (n < k) return false  
    for (i=1; i<=n; i++)  
        for (j=i+1; j<=n; i++)  
            if ( (c[i], c[j])  $\notin$  E ) return false  
    return true  
}
```

$O(v^2)$



# VERTEXCOVER กับ INDEPENDENTSET



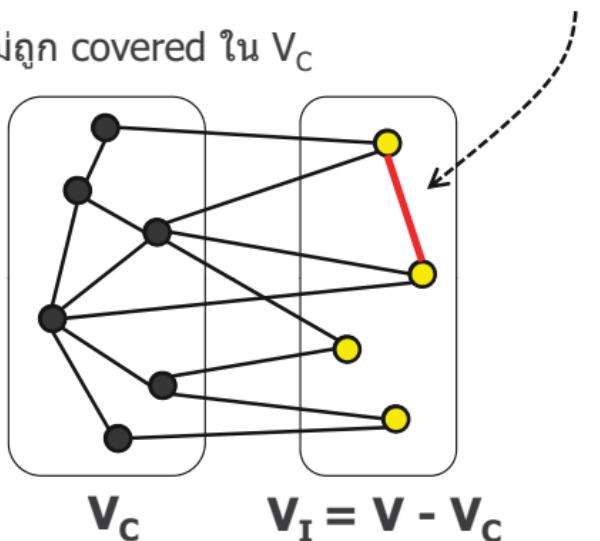
$\text{VC}(G, 3) \longleftrightarrow \text{IS}(G, 2-3)$

true  $\longleftrightarrow$  true

## VERTEX COVER กับ INDEPENDENT SET

ถ้า  $V_C$  เป็น vertex cover ของ  $G$ ,  $V - V_C$  เป็น independent set

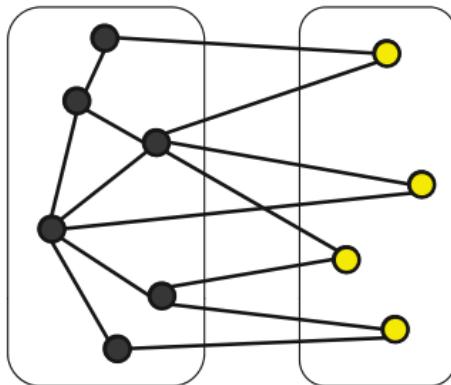
- ถ้า  $V_C$  เป็น vertex cover, ต้องไม่มีเส้นเชื่อมระหว่างปมใน  $V - V_C$
- เพราะถ้ามี จะไม่ถูก covered ใน  $V_C$



# VERTEX COVER กับ INDEPENDENT SET

ถ้า  $V_I$  เป็น independent set ของ  $G$ ,  $V - V_I$  เป็น vertex cover

- ถ้า  $V_I$  เป็น independent set, ไม่มีเส้นเชื่อมระหว่างปมของ  $V_I$
- ดังนั้นเส้นเชื่อมทุกเส้นต้องมีปมปลายอย่างน้อยหนึ่งปมอยู่ใน  $V - V_I$



$$V_C = V - V_I \quad V_I$$

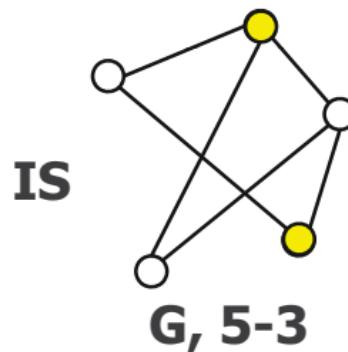
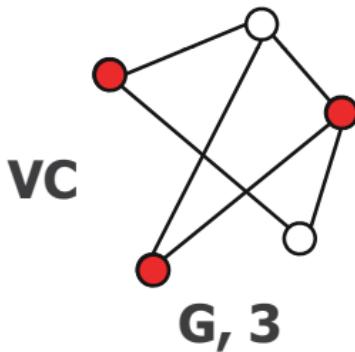
## VERTEX COVER กับ INDEPENDENT SET

$V_C$  เป็น vertex cover ของ  $G$   
ก็ต่อเมื่อ

$V - V_C$  เป็น independent set ของ  $G$

$G$  มี vertex cover ขนาด  $k$   
ก็ต่อเมื่อ

$G$  มี independent set ขนาด  $|V| - k$



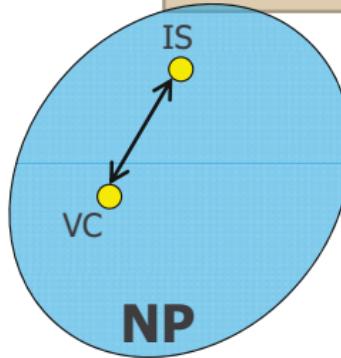
# Reductions

```
VertexCover( G=(V,E) , k ) {
    return IndependentSet( G, |V| - k )
}
```

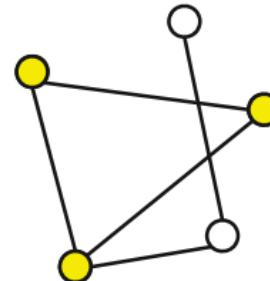
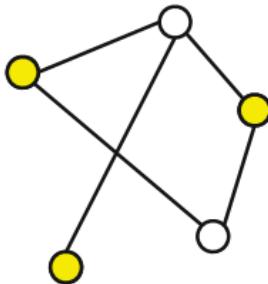
$\text{VERTEXCOVER} \leq_p \text{INDEPENDENTSET}$

```
IndependentSet( G=(V,E) , k ) {
    return VertexCover( G, |V| - k )
}
```

$\text{INDEPENDENTSET} \leq_p \text{VERTEXCOVER}$



## INDEPENDENT SET กับ CLIQUE



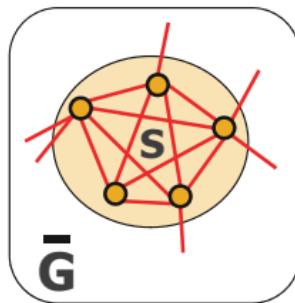
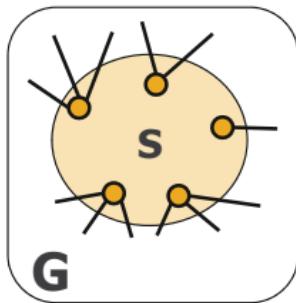
$\text{IS}(G, 3) \longleftrightarrow \text{CLIQUE}(\bar{G}, 3)$

true  $\longleftrightarrow$  true

# INDEPENDENT SET กับ CLIQUE

ถ้า  $S$  เป็น independent set ของ  $G$ ,  $S$  เป็น clique ของ  $\bar{G}$

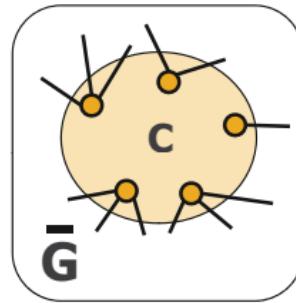
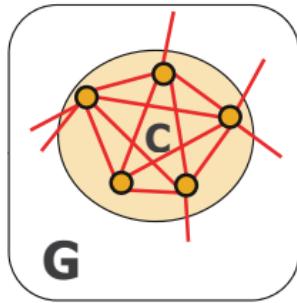
- ถ้า  $S$  เป็น independent set, ไม่มีเส้นเชื่อมระหว่างปมของ  $S$  ใน  $G$
- เมื่อสร้าง  $\bar{G}$  ต้องมีเส้นเชื่อมระหว่างทุกคู่ปมใน  $S$



## INDEPENDENT SET กับ CLIQUE

ถ้า  $C$  เป็น clique ของ  $G$ ,  $C$  เป็น independent set ของ  $\bar{G}$

- ถ้า  $C$  เป็น clique, มีเส้นเชื่อมระหว่างทุกคู่ปมของ  $C$  ใน  $G$
- ดังนั้นต้องไม่มีเส้นเชื่อมระหว่างคู่ปมใดใน  $C$  ของ  $\bar{G}$



## INDEPENDENT SET กับ CLIQUE

S เป็น independent set ของ G

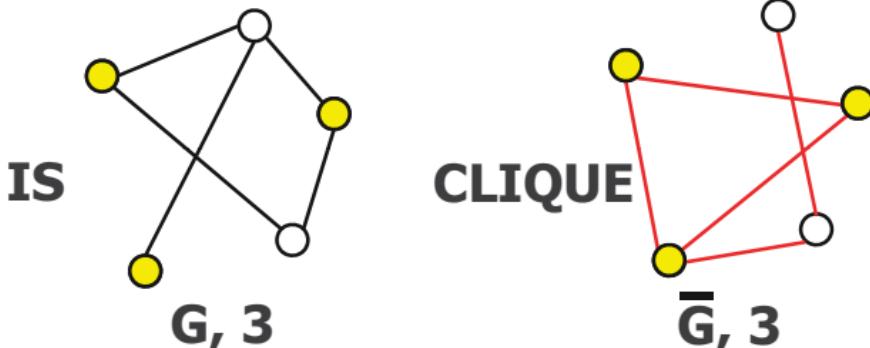
ก็ต่อเมื่อ

S เป็น clique ของ  $\bar{G}$

G มี independent set ขนาด k

ก็ต่อเมื่อ

$\bar{G}$  มี clique ขนาด k



# Reductions

```
IndependentSet( G=(V,E) , k ) {
```

$E'$  = เซตของเส้นเชื่อม  $(u,v)$  เมื่อไม่มี  $(u,v) \in E$

$G' = (V, E')$

```
return Clique( G' , k )
```

```
}
```

INDEPENDENTSET  $\leq_P$  CLIQUE

```
Clique( G=(V,E) , k ) {
```

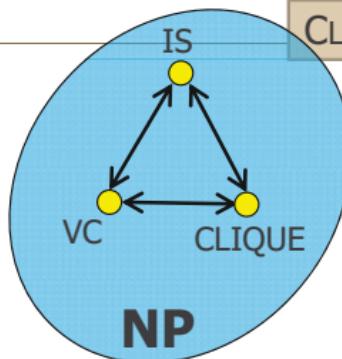
$E'$  = เซตของเส้นเชื่อม  $(u,v)$  เมื่อไม่มี  $(u,v) \in E$

$G' = (V, E')$

```
return IndependentSet( G' , k )
```

```
}
```

CLIQUE  $\leq_P$  INDEPENDENTSET



NP

# คงเขียนเอง

```
VertexCover( G=(V,E) , k ) {
    return Clique( )
}
```

$\text{VERTEXCOVER} \leq_p \text{ CLIQUE}$

```
Clique( G=(V,E) , k ) {
    return VertexCover( )
}
```

$\text{CLIQUE} \leq_p \text{ VERTEXCOVER}$

# NP-hard กับ NP-Complete

## ❖ X เป็นปัญหา NP-hard เมื่อ

❖ ทุกปัญหาใน NP ลดรูปไปเป็น X ได้ใน polynomial time

$$\forall q \in NP, q \leq_P X$$

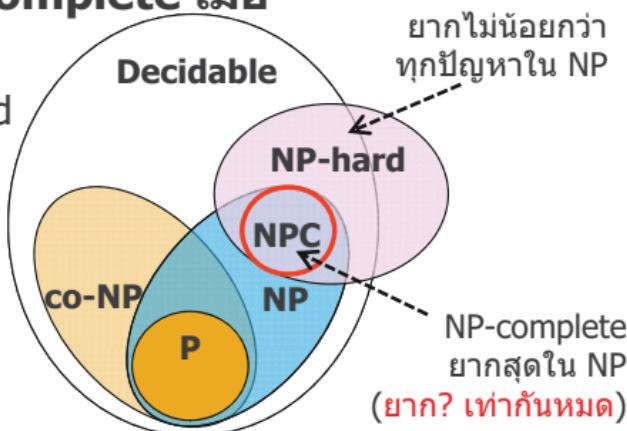
❖ NP-Hard ไม่ใช่ "ปัญหาที่อยู่ใน NP และเป็นปัญหายาก"

## ❖ X เป็นปัญหา NP-Complete เมื่อ

❖  $X \in NP$

❖ X เป็นปัญหา NP-hard

ขอตัวอย่างปัญหา  
ที่เป็น NP-hard



# Cook-Levin Theorem

- ❖ นิพจน์บูลีนในรูป  
**Product of Sum**  
**Conjunctive Normal Form**

SAT is NP-Complete

$$\begin{aligned} \text{❖ ต.ย. } f(x_1, x_2, x_3, x_4) &= (\bar{x}_1 + \bar{x}_2)(x_1 + x_2 + x_3)(\bar{x}_3 + \bar{x}_4) \\ &= (\bar{x}_1 \vee \bar{x}_2) \wedge (x_1 \vee x_2 \vee x_3) \wedge (\bar{x}_3 \vee \bar{x}_4) \end{aligned}$$

- ❖ truth assignment คือการให้ค่าจริง/เท็จกับ  $x_i$  ของ  $f$
- ❖ satisfying truth assignment คือ truth assignment ที่ทำให้  $f$  เป็นจริง  $x_1 = 0, x_2 = 0, x_3 = 1, x_4 = 0$

- ❖ ปัญหา SAT (**Satisfiability problem**)

- ❖ input : นิพจน์บูลีน  $f$  ในรูป Conjunctive Normal Form
- ❖ คำถาม :  $f$  มี satisfying truth assignment หรือไม่ ?

- ❖ **SAT ∈ NP**

- ❖ **SAT is NP-hard**

Proc. 3<sup>rd</sup> Annual ACM Symposium Theory of Computing  
May 1971

Complexity of Theorem-Proving Procedures

Stephen A. Cook

University of Toronto



Summary

It is shown that any recognition problem solved by a polynomial time-bounded nondeterministic Turing machine can be "reduced" to the problem of determining whether a given propositional formula is a tautology. Here "reduced" means, roughly speaking, that the first problem can be solved deterministically in polynomial time provided an oracle is available for solving the second. From this notion of reducible, polynomial degrees of difficulty are defined, and it is shown that the problem of determining tautologyhood has the same polynomial degree as the problem of determining whether the first of two given graphs is isomorphic to a subgraph of the second. Other examples are discussed. A method of measuring the complexity of proof procedures for the predicate calculus is introduced and discussed.

Throughout this paper, a set of

certain recursive set of strings on this alphabet, and we are interested in the problem of finding a good lower bound on its possible recognition times. We provide no such lower bound here, but theorem 1 will give evidence that {tautologies} is a difficult set to recognize, since many apparently difficult problems can be reduced to determining tautologyhood. By reduced we mean, roughly speaking, that if tautologyhood could be decided instantly (by an "oracle") then these problems could be decided in polynomial time. In order to make this notion precise, we introduce query machines, which are like Turing machines with oracles in [1].

A query machine is a multitape Turing machine with a distinguished tape called the query tape, and three distinguished states called the query state, yes state, and no state, respectively. If  $M$  is a query machine and  $T$  is a set of

## ПРОБЛЕМЫ ПЕРЕДАЧИ ИНФОРМАЦИИ

Том IX

1973

В  
каким  
могли  
и дру-  
гими  
доказа-  
дочам  
выше  
алгори-  
тмов  
букв  
более  
тов в  
доказа-  
дочам  
О  
стрем  
(в си-  
стем  
кими  
тами :  
Ф

Анал  
О  
будет  
с для  
прини-  
мати  
робот  
М  
рукот  
коопер  
А  
поддер  
сущес  
Д  
длитель  
ложни  
З  
скими  
Ф  
сущес  
Д  
З  
вие о  
тала. Задачи числа на границе и требуется продолжить их на всю матрицу с со-  
блуждением условия.

## КРАТКИЕ СООБЩЕНИЯ

Leonid Levin

УДК 519.14

## УНИВЕРСАЛЬНЫЕ ЗАДАЧИ ПЕРЕБОРА

*Л. А. Левин*

В статье рассматривается несколько известных массовых задач «переборного типа» и доказывается, что эти задачи можно решать лишь за такое время, за которое можно решать вообще любые задачи указанного типа.

После уточнения понятия алгоритма была доказана алгоритмическая неразрешимость ряда классических массовых проблем (например, проблем тождества элементов групп, гомеоморфности многообразий, разрешимости диофантовых уравнений и других). Тем самым был снят вопрос о нахождении практического способа их решения. Однако существование алгоритмов для решения других задач не снимает для них аналогичного вопроса из-за фантастически большого объема работы, предписываемого этими алгоритмами. Такова ситуация с так называемыми переборными задачами: минимизации булевых функций, поиска доказательств ограниченной длины, выяснения изоморфности графов и другими. Все эти задачи решаются тривиальными алгоритмами, состоящими в переборе всех возможностей. Однако эти алгоритмы требуют экспоненциального времени работы и у математиков сложилось убеждение, что



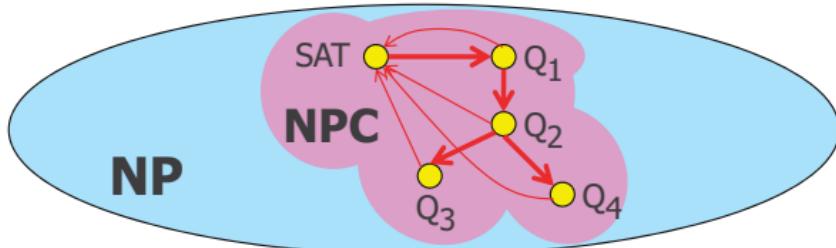
# การพิสูจน์ NP-Complete

## ❖ X เป็นปัญหา NP-Complete เมื่อ

- ❖  $X \in \text{NP}$
- ❖ X เป็นปัญหา NP-hard

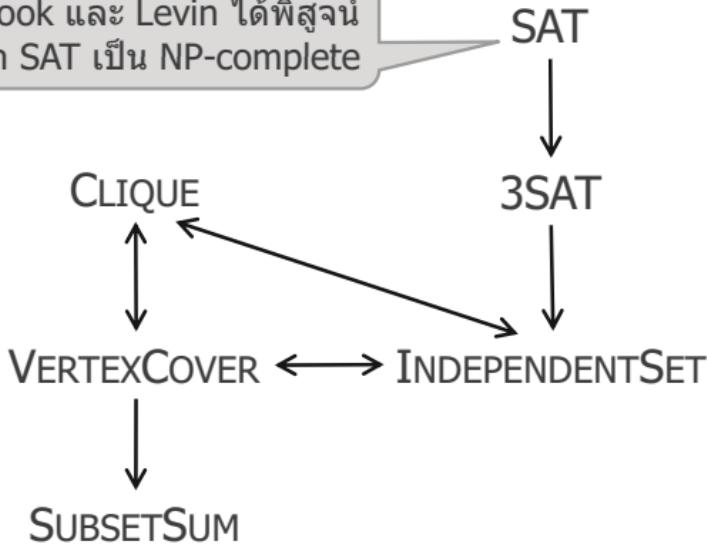
## ❖ X เป็น NP-hard

- ❖ ~~ทุกปัญหานอก NP ลดรูปไปเป็น X ได้ใน polynomial time~~
- ❖ เลือกปัญหา Q ที่รู้แล้วว่าเป็น NP-hard,  
แล้วแสดงให้เห็นว่า Q ไม่ยากกว่า X,  
(Q ลดรูปเป็น X ได้,  $Q \leq_p X$ )



# ตัวอย่าง

Cook และ Levin ได้พิสูจน์ว่า SAT เป็น NP-complete



# 3SAT is NP-Complete

- ❖ 3SAT คือ SAT ที่แต่ละ clause มีได้แค่ 3 literals

$$(\bar{x}_1 + x_2 + x_3) (x_1 + \bar{x}_2 + x_4) (\bar{x}_1 + \bar{x}_4 + \bar{x}_3)$$

- ❖ 3SAT  $\in$  NP

- ❖ 3SAT is NP-hard

- ❖ เลือก SAT ที่รู้แล้วว่า เป็น NP-Complete
- ❖ จะแสดงให้เห็นว่า  $SAT \leq_p 3SAT$   
(SAT is polynomially reducible to 3SAT)



- ❖  $SAT \leq_p 3SAT$

- ❖ วิธีแปลงนิพจน์บูลีนของ SAT เป็น นิพจน์บูลีนของ 3SAT
- ❖ ทำให้ค่าตอบ Yes/No ของ 3SAT เป็นค่าตอบของ SAT

# Reduction : SAT → 3SAT

SAT  
 $(x_1)(x_2 + \bar{x}_3)(x_1 + x_3 + \bar{x}_4)(x_1 + \bar{x}_2 + x_3 + x_4 + x_5)$



3SAT

$$(x_1 + \bar{s}_1 + \bar{r}_1)(x_1 + \bar{s}_1 + r_1)(x_1 + s_1 + \bar{r}_1)(x_1 + s_1 + r_1)$$

$$(x_2 + \bar{x}_3 + \bar{p}_1)(x_2 + \bar{x}_3 + p_1)$$

$$(x_1 + x_3 + \bar{x}_4)$$

$$(x_1 + \bar{x}_2 + q_1)(\bar{q}_1 + x_3 + q_2)(\bar{q}_2 + x_4 + x_5)$$

## การแปลง clause ที่มี 2 ตัวแปร

$$(x_2 + \bar{x}_3) = (x_2 + \bar{x}_3 + \bar{p}_1)(x_2 + \bar{x}_3 + p_1)$$

$$\begin{aligned} & ((x_2 + \bar{x}_3) + \bar{p}_1)((x_2 + \bar{x}_3) + p_1) \\ &= (x_2 + \bar{x}_3) + (x_2 + \bar{x}_3)p_1 + (x_2 + \bar{x}_3)\bar{p}_1 + (\bar{p}_1p_1) \\ &= (x_2 + \bar{x}_3) + (x_2 + \bar{x}_3)(p_1 + \bar{p}_1) + (\bar{p}_1p_1) \\ &= (x_2 + \bar{x}_3) + (x_2 + \bar{x}_3)(1) + 0 \\ &= (x_2 + \bar{x}_3) \end{aligned}$$

จำนวน clauses เพิ่มขึ้น 1, ตัวแปรเพิ่มอีก 1

# การแปลง clause ที่มี 1 ตัวแปร

$$(x_1) = (x_1 + (\bar{s}_1 + \bar{r}_1))(x_1 + (\bar{s}_1 + r_1))$$

$$(x_1 + (s_1 + \bar{r}_1))(x_1 + (s_1 + r_1))$$

$$(x_1 + (\bar{s}_1 + \bar{r}_1))(x_1 + (\bar{s}_1 + r_1))(x_1 + (s_1 + \bar{r}_1))(x_1 + (s_1 + r_1))$$

$$= x_1 + x_1((\bar{s}_1 + \bar{r}_1) + (\bar{s}_1 + r_1) + (s_1 + \bar{r}_1) + (s_1 + r_1)) +$$

$$(\bar{s}_1 + \bar{r}_1)(\bar{s}_1 + r_1)(s_1 + \bar{r}_1)(s_1 + r_1)$$

$$= x_1 + x_1(1) + 0$$

$$= x_1$$

จำนวน clauses เพิ่มขึ้น 3, ตัวแปรเพิ่มอีก 2

## การแปลง clause ที่มี > 3 ตัวแปร

$$\begin{aligned} & (x_1 + \bar{x}_2 + x_3 + \bar{x}_4 + x_5 + \bar{x}_6 + \bar{x}_7) \\ &= (x_1 + \bar{x}_2 + q_1) \\ & (\bar{q}_1 + x_3 + q_2) \\ & (\bar{q}_2 + \bar{x}_4 + q_3) \\ & (\bar{q}_3 + x_5 + q_4) \\ & (\bar{q}_4 + \bar{x}_6 + \bar{x}_7) \end{aligned}$$

ตัวแปรมี k ตัว  $\rightarrow$  จำนวน clauses เพิ่มขึ้น k-2, ตัวแปรเพิ่มอีก k-3

# การแปลง clause ที่มี > 3 ตัวเป็น

$$\begin{aligned}
 & (x_1 + \bar{x}_2 + x_3 + \bar{x}_4 + x_5 + \bar{x}_6 + \bar{x}_7) \\
 = & (x_1 + \bar{x}_2 + q_1) \\
 & (\bar{q}_1 + x_3 + q_2) \\
 & (\bar{q}_2 + \bar{x}_4 + q_3) \\
 & (\bar{q}_3 + x_5 + q_4) \\
 & (\bar{q}_4 + \bar{x}_6 + \bar{x}_7)
 \end{aligned}$$

instance ของ SAT แปลงไปเป็น instance ของ 3SAT  
ที่มีขนาดเป็น polynomial ของขนาดของ SAT

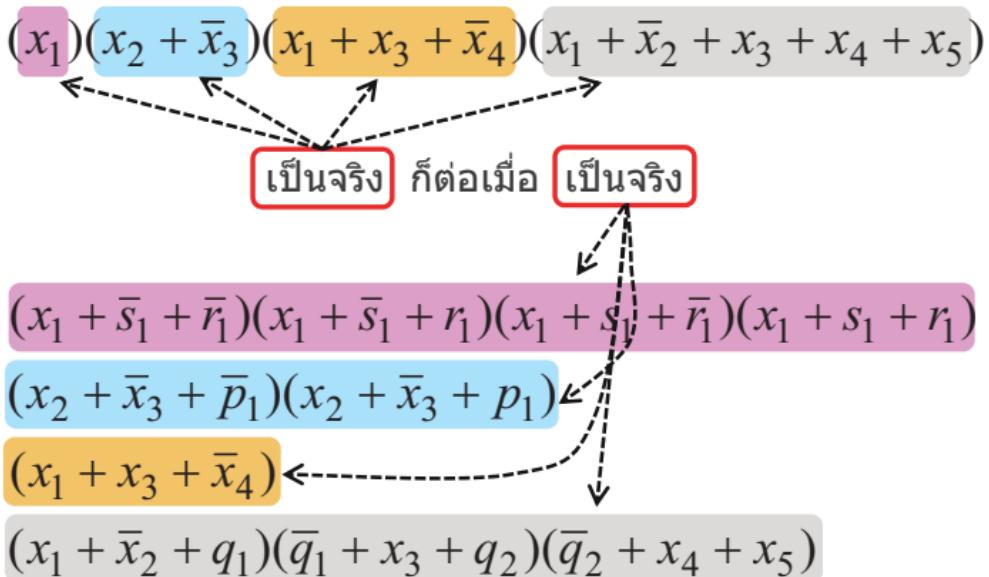
# Reduction : SAT → 3SAT

SAT  
 $(x_1)(x_2 + \bar{x}_3)(x_1 + x_3 + \bar{x}_4)(x_1 + \bar{x}_2 + x_3 + x_4 + x_5)$

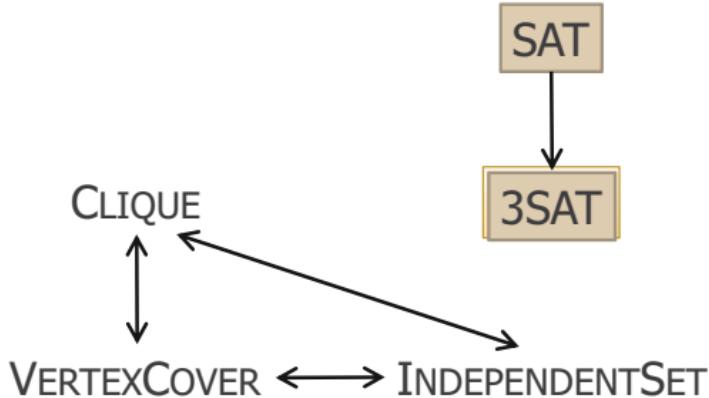


$(x_1 + \bar{s}_1 + \bar{r}_1)(x_1 + \bar{s}_1 + r_1)(x_1 + s_1 + \bar{r}_1)(x_1 + s_1 + r_1)$   
 $(x_2 + \bar{x}_3 + \bar{p}_1)(x_2 + \bar{x}_3 + p_1)$   
 $(x_1 + x_3 + \bar{x}_4)$   
 $(x_1 + \bar{x}_2 + q_1)(\bar{q}_1 + x_3 + q_2)(\bar{q}_2 + x_4 + x_5)$

# Reduction : SAT → 3SAT



# ตัวอย่าง



SUBSETSUM

# ลองคิดดู : ยากไหม ?

- ❖ **1SAT :**  $(x_1)(x_2)(\bar{x}_3)(\bar{x}_4)(x_5)(\bar{x}_6)$
- ❖ **2SAT :**  $(x_1 + \bar{x}_2)(x_1 + \bar{x}_3)(x_2 + \bar{x}_4)(x_3 + x_4)$
- ❖ หา **satisfying truth assignment** ของนิพจน์  
บูลีนในรูป **sum of product** หรือ **disjunctive normal form**

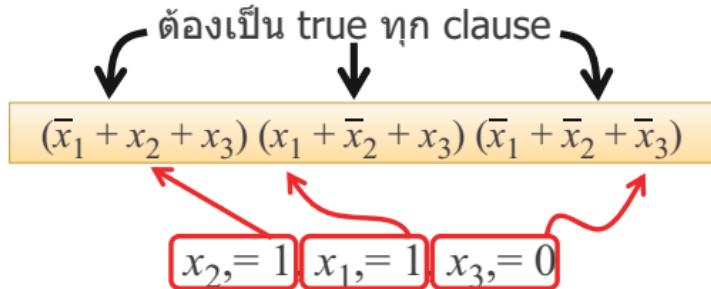
$$(x_1\bar{x}_2x_4) + (x_1x_2\bar{x}_3) + (x_2\bar{x}_4) + (x_1\bar{x}_2x_3x_4)$$

# INDEPENDENTSET is NP-Complete

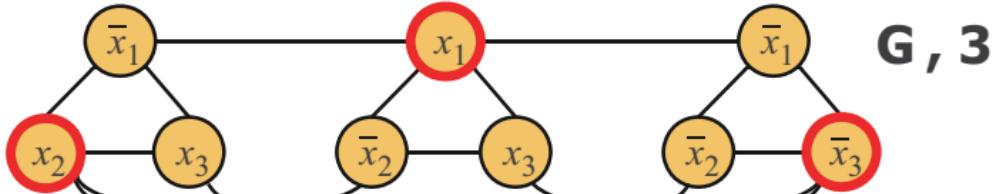
- ❖ **INDEPENDENTSET ∈ NP**
- ❖ **INDEPENDENTSET is NP-hard**
  - ❖ เลือก 3SAT ที่รู้แล้วว่า เป็น NP-Complete
  - ❖ จะแสดงให้เห็นว่า  $3SAT \leq_p INDEPENDENTSET$
- ❖  **$3SAT \leq_p INDEPENDENTSET$** 
  - ❖ วิธีแปลงนิพจน์บูลีนของ 3SAT เป็นกราฟ G และจำนวน k
  - ❖ คำตอบ Yes/No ของ INDEPENDENTSET เป็นคำตอบของ 3SAT



# $3SAT \leq_p INDEPENDENTSET$



สร้างกราฟ  $G$  มี 1 สามเหลี่ยมแทน 1 clause  
มีเส้นเชื่อมข้ามสามเหลี่ยมระหว่างปมที่ "ขัด" กัน  
หา  $INDEPENDENTSET(G, k)$   $k$  คือจำนวน clauses



## **3SAT $\leq_p$ INDEPENDENTSET**

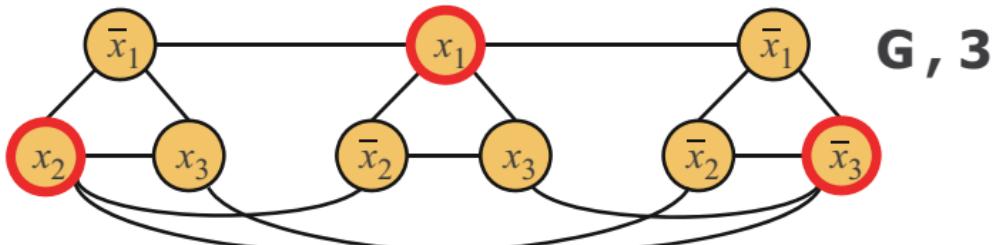
ต้องเป็น true ทุก clause

$$(\bar{x}_1 + x_2 + x_3) (\bar{x}_1 + \bar{x}_2 + x_3) (\bar{x}_1 + \bar{x}_2 + \bar{x}_3)$$

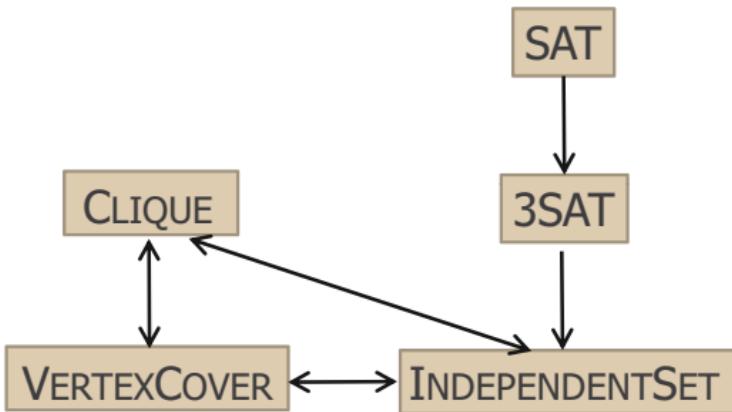
$x_2 = 1, x_1 = 1, x_3 = 0$

INDEPENDENTSET( $G, k$ ) คืนจริง  $\rightarrow$  3SAT( $f$ ) เป็นจริง

3SAT( $f$ ) เป็นจริงได้  $\rightarrow$  INDEPENDENTSET( $G, k$ ) ต้องคืนจริง



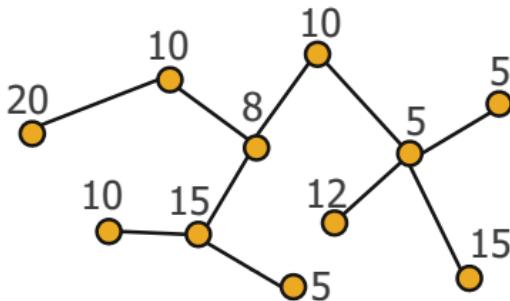
# ตัวอย่าง



SUBSETSUM

# ลองคิดดู : ยากไหม ?

ถ้าต้องการ Independent Set  
ขนาดใหญ่สุดของต้นไม้



ถ้าแต่ละปมมีน้ำหนัก และต้องการ Independent Set ของต้นไม้ที่มีน้ำหนักปมรวมในเขตมากสุด

มี Independent Set ของต้นไม้ที่มีน้ำหนักปมรวมในเขตอย่างน้อย  $k$  หรือไม่

# SUBSETSUM is NP-Complete

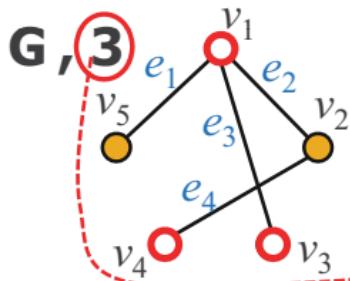
- ❖ **SUBSETSUM ∈ NP**
- ❖ **SUBSETSUM is NP-hard**
  - ❖ เลือก VERTEXCOVER ที่รู้แล้วว่า เป็น NP-Complete
  - ❖ จะแสดงให้เห็นว่า  $\text{VERTEXCOVER} \leq_p \text{SUBSETSUM}$
- ❖  **$\text{VERTEXCOVER} \leq_p \text{SUBSETSUM}$** 
  - ❖ วิธีแปลง  $G, k$  ของ VERTEXCOVER เป็น  $D, m$  ของ SUBSETSUM
  - ❖ คำตอบ Yes/No ของ SUBSETSUM เป็นของ VERTEXCOVER ด้วย



# VERTEXCOVER $\leq_p$ SUBSETSUM

	$e_1$	$e_2$	$e_3$	$e_4$
$v_1$	1	1	1	0
$v_2$	0	1	0	1
$v_3$	0	0	1	0
$v_4$	0	0	0	1
$v_5$	1	0	0	0

incidence matrix



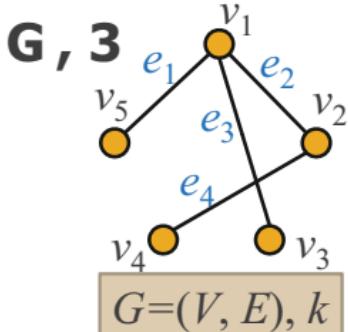
	$e_1$	$e_2$	$e_3$	$e_4$	$D$
$d_1 =$	1	1	1	0	11110
$d_2 =$	1	0	1	1	10101
$d_3 =$	1	0	0	1	10010
$d_4 =$	1	0	0	1	10001
$d_5 =$	1	1	0	0	11000
$d_6 =$	0	0	0	1	1
$d_7 =$	0	0	0	1	10
$d_8 =$	0	0	1	0	100
$d_9 =$	0	1	0	0	1000
$m =$					32222

เลข 2 ทุกหลัก

# VERTEXCOVER $\leq_p$ SUBSETSUM

	$e_1$	$e_2$	$e_3$	$e_4$
$v_1$	1	1	1	0
$v_2$	0	1	0	1
$v_3$	0	0	1	0
$v_4$	0	0	0	1
$v_5$	1	0	0	0

incidence matrix



	$e_1$	$e_2$	$e_3$	$e_4$	$D$
$d_1 =$	1	1	1	0	11110
$d_2 =$	1	0	1	0	10101
$d_3 =$	1	0	0	1	10010
$d_4 =$	1	0	0	0	10001
$d_5 =$	1	1	0	0	11000
$d_6 =$	0	0	0	0	1
$d_7 =$	0	0	0	1	10
$d_8 =$	0	0	1	0	100
$d_9 =$	0	1	0	0	1000

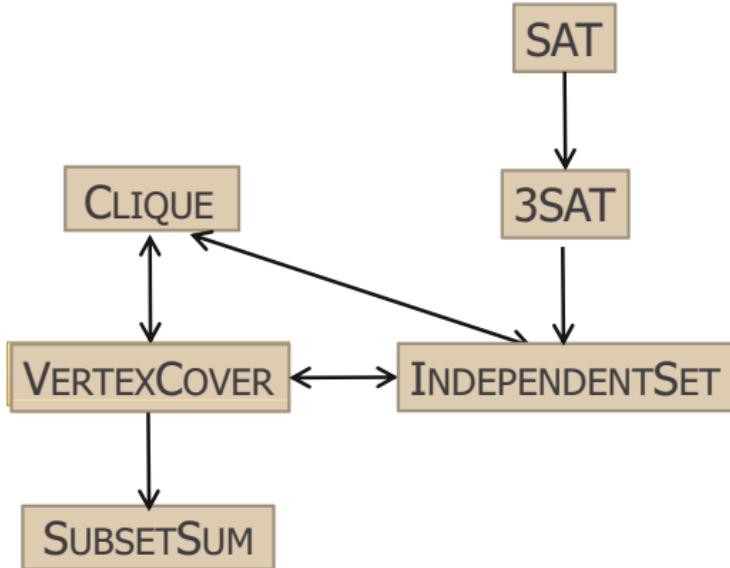
$$n = |V| + |E|$$

$$D = \{d_1, d_2, \dots, d_n\}$$

$$m = (k22\dots2)_{10}$$

$$m = 32222$$

# ตัวอย่าง

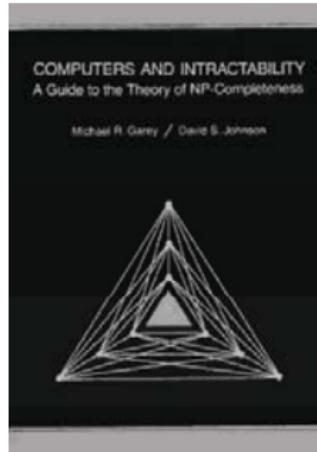


# ลองทำดู

**SAT  $\leq_p$  CLIQUE**

# ปัญหา NP-Complete มีมาก

- ❖ หนังสือฉบับตำนานของ Garey & Johnson, 1979
- ❖ รวบรวมหลายร้อยปัญหาที่เป็น NP-complete
- ❖ นำเสนอแนวทางการลดรูปและเอกสารอ้างอิง



# P = NP ?

- ❖ ถ้ามีโครงสร้างพื้นฐาน polynomial-time algo. สำหรับ ปัญหานึง ที่เป็น NP-complete
- ❖ เราจะได้ polynomial-time algo. สำหรับ ทุกปัญหา ที่เป็น NP-complete
- ❖ นั่นคือ ทุกปัญหาใน NP หาคำตอบได้ใน polynomial time (เป็น tractable problem)
- ❖ จะสรุปได้ว่า P = NP
- ❖ แต่ จนถึงปัจจุบัน ก็ยังไม่มีโครงทำได้สำเร็จ



## **P ≠ NP ?**

- ❖ ถ้ามีใครสักคน พิสูจน์ได้ว่า ต้องใช้เวลาอย่าง  
น้อยเป็น exponential ใน การหาคำตอบของ  
ปัญหานั่นในกลุ่มปัญหา NP-complete
- ❖ จะสรุปได้ว่า ปัญหา NP-complete ทุกปัญหา  
เป็น intractable problem
- ❖ นั่นคือ  $P \neq NP$
- ❖ แต่ จนถึงปัจจุบัน ก็ยังไม่มีใครทำได้สำเร็จ



## ลองคิดดู

- ❖ การพิสูจน์ว่า  $P = NP$  กับการพิสูจน์ว่า  $P \neq NP$  แบบใดสะเทือนวงการ ?
- ❖ จริงหรือไม่
  - ❖ ปัญหา  $P = NP$  ? เป็น decision problem
  - ❖ ปัญหา  $P = NP$  ? เป็นปัญหาในกลุ่ม NP
- ❖ ถ้าปัญหา  $P = NP$  ? เป็นปัญหา NP-complete หมายความว่าอย่างไร ?

Millennium Prize Problems - Mozilla Firefox

File Edit View History Bookmarks Tools Help

CM http://www.claymath.org/millennium/ Leonid Levin

Millennium Prize Problems

# Clay Mathematics Institute

Dedicated to increasing and disseminating mathematical knowledge



HOME ABOUT CMI PROGRAMS NEWS & EVENTS AWARDS SCHOLARS PUBLICATIONS

## Millennium Problems

In order to celebrate mathematics in the new millennium, The Clay Mathematics Institute of Cambridge, Massachusetts (CMI) has named seven *Prize Problems*. The Scientific Advisory Board of CMI selected these problems, focusing on important classic questions that have resisted solution over the years. The Board of Directors of CMI designated a \$7 million prize fund for the solution to these problems, with \$1 million allocated to each. During the Millennium Meeting held on May 24, 2000 at the Collège de France, Timothy Gowers presented a lecture entitled *The Importance of Mathematics*, aimed for the general public, while John Tate and Michael Atiyah spoke on the problems. The CMI invited specialists to formulate each problem.

One hundred years earlier, on August 8, 1900, David Hilbert delivered his famous lecture about open mathematical problems at the second International Congress of Mathematicians in Paris. This influenced our decision to announce the millennium problems as the central theme of a Paris meeting.

The rules for the award of the prize have the endorsement of the CMI Scientific Advisory Board and the approval of the Directors. The members of these boards have the responsibility to preserve the nature, the integrity, and the spirit of this prize.

[Birch and Swinnerton-Dyer Conjecture](#)

[Hodge Conjecture](#)

[Navier-Stokes Equations](#)

[P vs NP](#)

[Poincaré Conjecture](#)

[Riemann Hypothesis](#)

[Yang-Mills Theory](#)

[Rules](#)

[Millennium Meeting Videos](#)

# อัลกอริทึมสำหรับ ปัญหา NP-Hard

สมชาย ประสิทธิ์จุตระกุล

มหาวิทยาลัยเทคโนโลยีราชมงคลรัตนโกสินทร์

# ปัญหา NP-complete นั้นยาก

รูปจากหนังสือของ Garey & Johnson

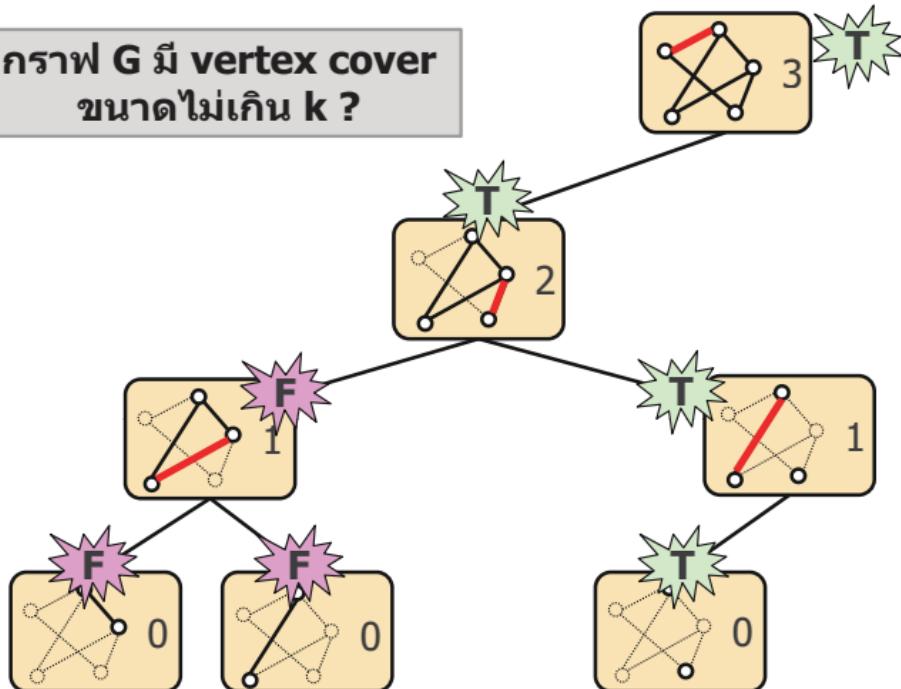


## พิจารณาตัวปัญหาให้รอบคอบ

- ❖ **ปัญหาของเราเป็นกรณีเฉพาะหรือไม่**
    - ❖ SAT ยาก แต่ 1SAT, 2SAT ง่าย
    - ❖ Independent Set ของกราฟยาก แต่ของตันไม่ง่าย
    - ❖ 3-Color ยาก แต่ 2-Color ง่าย
    - ❖ ...
  - ❖ บาง input ของปัญหาของเรามีขนาดเล็ก หรือไม่

# VertexCover(G, k) เมื่อ k มีค่าน้อย

กราฟ G มี vertex cover  
ขนาดไม่เกิน k ?



# Vertex Cover

```

hasVertexCover( G=(V,E) , k ) {
    if (|E| == 0) return true
    if (k == 0) return false
    choose an edge (u,v) ∈ E
    if ( hasVertexCover( G-{u} , k-1 ) OR
        hasVertexCover( G-{v} , k-1 ) )
        return true
    else
        return false
}

```

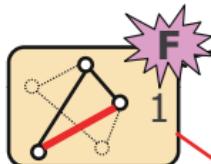
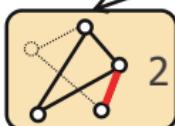
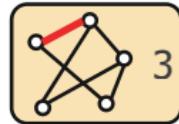
 $O(2^k(n+m))$ 

ถ้ารู้ว่า  $k$  มีค่าน้อย, ถึงแม้กราฟใหญ่ ก็ยังเร็ว  
หรือถ้า  $k$  เป็นค่าคงตัว เราได้ polynomial-time algorithm

**FPT : Fixed Parameter Tractable**

# ปรับปรุง

กราฟ G มี vertex cover  
ขนาดไม่เกิน k ?



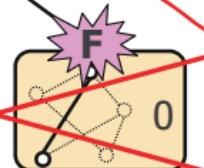
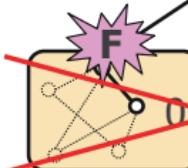
ข้อสังเกต :

ถ้า G มี n ปม, แต่ละปมนี้ degree ไม่เกิน  $n-1$

ถ้า G มี vertex cover ขนาดไม่เกิน k ปม

G ต้องมีเส้นเชื่อมไม่เกิน  $k(n-1)$  เส้น

$$3 \text{ เส้น} > 1 \times (3-1) = 2 \text{ เส้น}$$



# อัลกอริทึมสำหรับ Vertex Cover

```
hasVertexCover( G=(V,E) , k ) {  
    if (|E| == 0) return true  
    if (|E| > k(|V|-1)) return false  
    choose an edge (u,v) ∈ E  
    if ( hasVertexCover( G-{u} , k-1 ) OR  
        hasVertexCover( G-{v} , k-1 ) )  
        return true  
    else  
        return false  
}
```

ลองปรับปรุงให้เป็น

อัลกอริทึมเพื่อ~~หา~~ vertex cover ขนาด  $k$  ของกราฟ  $G$

# อัลกอริทึมสำหรับปัญหา NP-hard

## ❖ Backtracking, branch and bound

- ❖ ได้คำตอบที่ "ดีสุด" แต่อาจช้า
- ❖ อย่าลืมว่า ไม่จำเป็นว่าทุก instance ของปัญหา NP-hard ต้องใช้เวลานานในการหาคำตอบ

## ❖ Local search

- ❖ ได้คำตอบที่ "ดีมาก" ในทางปฏิบัติ แต่ไม่เสมอไป

## ❖ Approximation

- ❖ ได้คำตอบที่ "ดี" (ประกันคุณภาพ) และเร็วด้วย

# NP-Complete $\neq$ ยากทุก instances

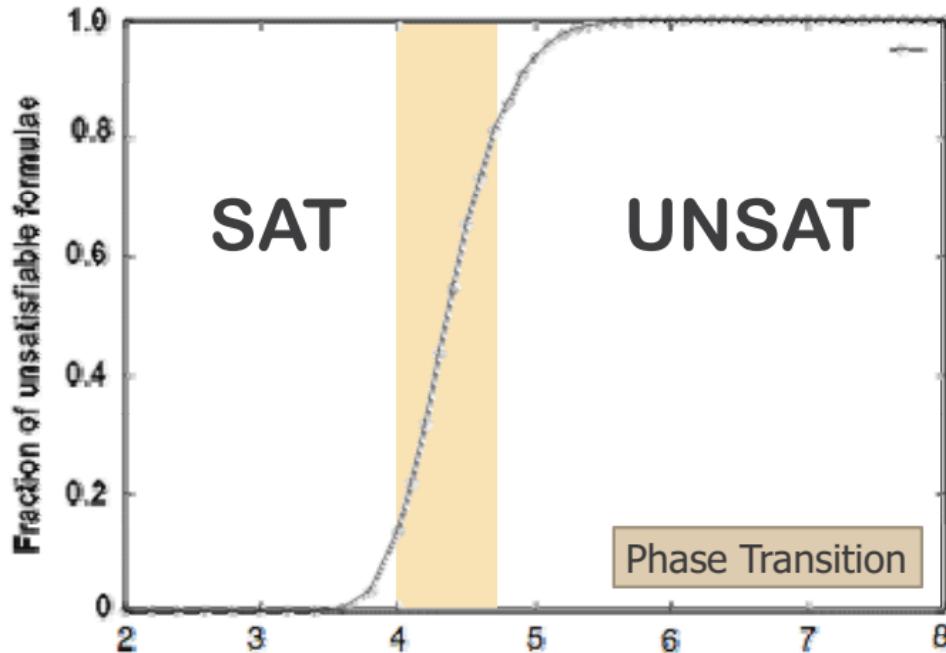
$$(x_1 + \bar{x}_2 + \bar{x}_3 + x_5 + \bar{x}_7)(x_2 + x_3 + x_4 + \bar{x}_6 + \bar{x}_7)$$

#clauses น้อย #ตัวแปรมาก นำจะ satisfy ได้ง่าย

$$(x_1 + \bar{x}_2)(x_1 + x_2 + x_3)(\bar{x}_1 + x_2 + x_3)(\bar{x}_2 + \bar{x}_3)(x_3 + \bar{x}_2)(x_1 + \bar{x}_3)$$

#clauses มาก #ตัวแปรน้อย นำจะ satisfy ไดยาก

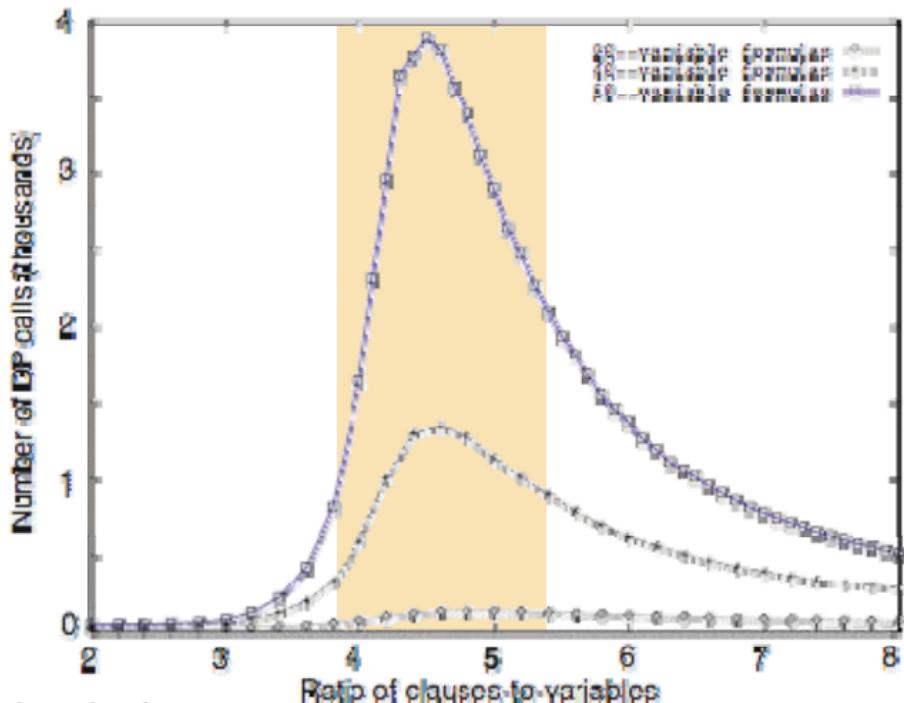
# NP-Complete ≠ ยากทุก instances



Kirkpatrick and Selman

Ratio of clauses-to-variables

# NP-Complete $\neq$ ยากทุก instances



Kirkpatrick and Selman



# Approximation Algorithms

# Approximation Algorithms

- ❖ อัลกอริทึมที่ทำงานใน **polynomial time**
- ❖ ใช้กับ **NP-hard optimization problems**
- ❖ **ประกันคุณภาพว่า**
  - ❖ ผลที่ได้มีคุณภาพไม่เลวกว่า  $\delta$  เท่าของคำตอบที่ดีสุด
  - ❖ ต้องพิสูจน์การประกันคุณภาพ

การเดินทางของพนักงานขาย  
ที่มีระยะทางรวม ไม่เกิน 2 เท่า  
ของการเดินทางที่สั้นสุด  
(2-approx.)



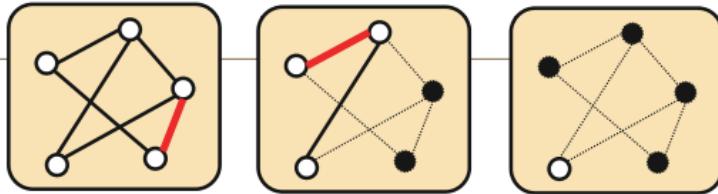
# Vertex Cover

- ❖ หา vertex cover ที่มีขนาดเล็กสุดเป็น NP-hard
- ❖ approx. : หาที่มีขนาดไม่เกิน 2 เท่าของที่เล็กสุด

```

approxVertexCover( G=(V,E) ) {
    Vc = {}
    G* = G
    while ( กราฟ G* ยังมีเส้นเชื่อม ) {
        (u,v) = เลือกเส้นเชื่อมของ G* มากที่สุด
        Vc = Vc ∪ {u, v}
        G* = G* - {u, v} // ลบเส้นเชื่อมที่ติดกับ u, v ออกด้วย
    }
    return Vc
}

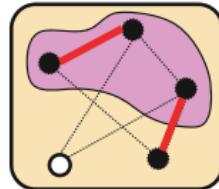
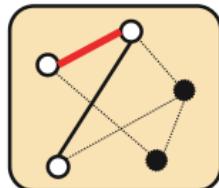
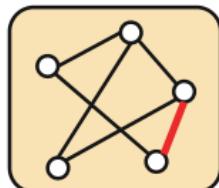
```



## Vertex Cover : 2-approx.

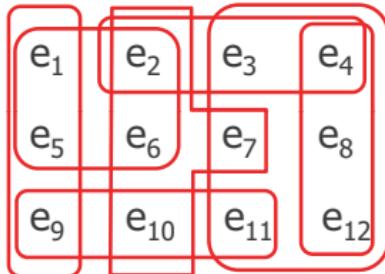
- ❖ ให้  $E^*$  คือเซตของเส้นเชื่อมที่ถูกเลือก
  - ❖ เส้นเชื่อมใน  $E^*$  ต้องไม่มีจุดปลายร่วมกัน
  - ❖  $V_c$  เก็บจุดปลายของเส้นเชื่อมใน  $E^*$
  - ❖  $|V_c| = 2|E^*|$
- ❖ ให้  $V_{OPT}$  คือ optimal vertex cover
  - ❖ เส้นเชื่อม  $e \in E^*$  ต้องมีจุดปลายใน  $V_{OPT}$
  - ❖ ถ้า  $v_1 \in V_{OPT}$  เป็นจุดปลายของ  $e_1 \in E^*$   
 $v_1$  ต้องไม่เป็นจุดปลายของเส้นอื่นใน  $E^*$
  - ❖ ดังนั้น  $|E^*| \leq |V_{OPT}|$
- ❖ สรุปได้ว่า  $|V_c| \leq 2|V_{OPT}|$

2-approx.



# Set Cover

- ❖ **Input :** เซต  $U$ , และ  $s_1, s_2, \dots, s_m \subseteq U$
- ❖ **Output :** กลุ่มที่มีขนาดเล็กสุดของเซตย่อย  $s_k$  ที่มีสมาชิกรวมเป็น  $U$
- ❖ หาก **set cover** ขนาดเล็กสุดเป็น NP-hard
- ❖ **approx.** : หากมีขนาดไม่เกิน  $\ln n$  เท่าของ  $OPT$   
 $n$  คือจำนวนสมาชิกของ  $U$



$\text{VERTEXCOVER} \leq_p \text{SETCOVER}$

# Approx. Set Cover

```
approxSetCover( U, S[1..m] ) {  
    CApprox = {}  
    while ( U ≠ ∅ ) {  
        sMAX = เลือกเซตใน S ที่มีสมาชิกปรากฏใน U มากสุด  
        CApprox = CApprox ∪ {sMAX}  
        U = U - sMAX  
    }  
    return CApprox  
}
```

- ❖ ให้  $C_{Opt}$  แทน set cover ขนาดเล็กสุด
- ❖ ให้  $C_{Approx}$  แทน set cover จาก approx. algo.
- ❖ ให้  $n$  คือจำนวนสมาชิกของ  $U$
- ❖ จะแสดงให้เห็นว่า  $|C_{Approx}| / |C_{Opt}| = \ln n$

# Approx. Set Cover : Analysis

## ❖ หลังรอบที่ $k$ :

- ❖ ให้  $n_k$  คือจำนวนสมาชิกที่เหลือใน  $U$  ที่ยังไม่ถูก covered
- ❖ สมาชิกที่เหลือนี้ต้องถูก covered ด้วย  $C_{OPT}$  ได้
- ❖ เขตย่อยที่ cover สมาชิกที่เหลือของ  $U$  ได้มากสุด ต้องมีขนาดอย่างน้อย  $n_k / |C_{Opt}|$  ตัว

$$n_k = 20, |C_{opt}| = 5$$

## ❖ หลังรอบที่ $k+1$ :

$$n_{k+1} \leq n_k \left(1 - \frac{1}{|C_{Opt}|}\right)$$

$$n_k \leq n_{k-1} \left(1 - \frac{1}{|C_{Opt}|}\right) \leq n_{k-2} \left(1 - \frac{1}{|C_{Opt}|}\right)^2 = \dots$$

$$\frac{|C_{Approx}|}{|C_{Opt}|} = \ln n$$

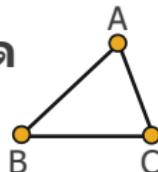
$$\leq n_0 \left(1 - \frac{1}{|C_{Opt}|}\right)^k \leq n_0 \left(e^{-1/|C_{Opt}|}\right)^k = n e^{-k/|C_{Opt}|} < 1, \text{ เมื่อ } \frac{k}{|C_{Opt}|} = \ln n$$

$$(1-x) \leq e^{-x}$$

$$|C_{Approx}| = \text{จำนวนรอบที่ทำงาน}$$

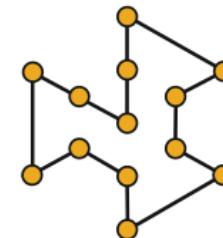
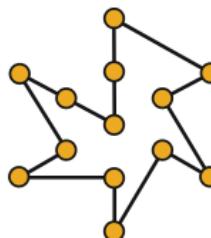
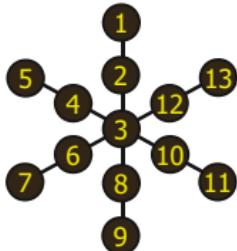
# Metric TSP

- ❖ หา tour ที่สั้นสุดเป็น NP-hard
- ❖ approx : หาที่ยาวไม่เกิน 2 เท่าของที่สั้นสุด
- ❖ ใช้กับระยะทางแบบ  $d_{AB} + d_{BC} \geq d_{AC}$

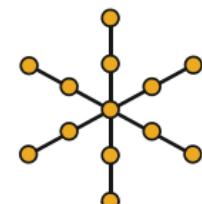
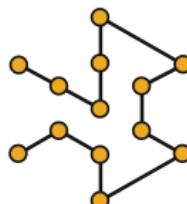


```
approxEuclideanTSP ( G=(V,E) ) {  
     $T_{MST} = \text{MinimumSpanningTree}(G)$   
    tour = PreorderTraversal(  $T_{MST}$  )  
    return tour  
}
```

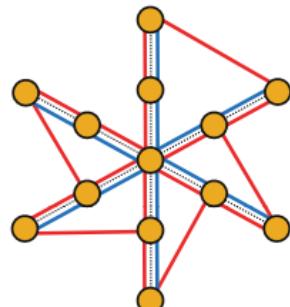
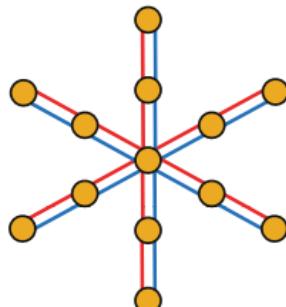
$$L_{\text{Approx}} \leq 2 L_{\text{Opt}}$$



# 2-Approx. TSP



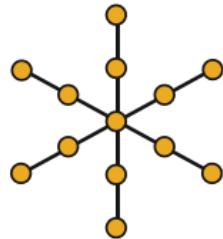
$$L_{Opt} > L \geq L_{MST}$$



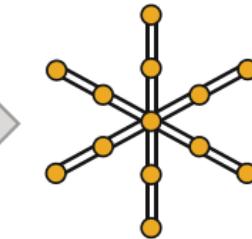
$$2L_{Opt} > 2L_{MST} \geq$$

$$L_{Approx} < 2L_{Opt}$$

# 1.5-Approx. TSP



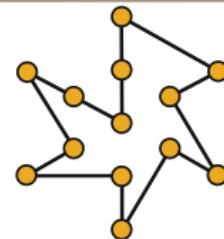
หา MST



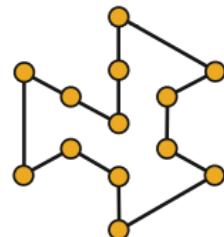
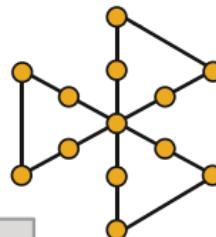
สร้าง Euler graph



$$L_{\text{Approx}} < 2L_{\text{Opt}}$$



หา tour ไม่ผ่านปมซ้ำ  
โดยใช้ทางลัด

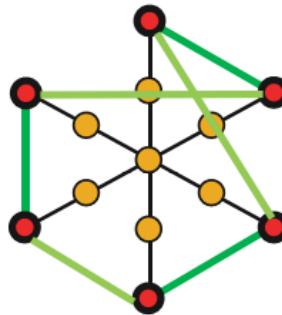


$$L_{\text{Approx}} < 1.5L_{\text{Opt}}$$

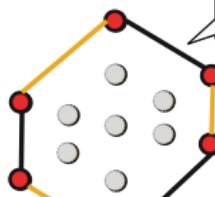
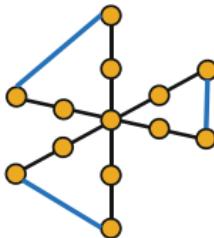
ทุกปมใน Euler graph  
ต้องมี degree เป็นจำนวนคู่

# สร้าง Euler Graph ที่ดีกว่า

- ❖ ให้  $V_{Odd}$  คือเซตของปมใน MST ที่มีดีกรีเป็นจำนวนคี่
  - ❖ หมายเหตุ :  $|V_{Odd}|$  เป็นจำนวนคู่
- ❖ หาเส้นเชื่อมที่จับทุกคู่ปมใน  $V_{Odd}$  ที่มีความยาวเส้นเชื่อมรวมน้อยสุด (minimal length matching)
- ❖ เพิ่มเส้นเชื่อมจับคู่  $V_{Odd}$  ใน MST ได้กราฟอยู่เลอร์



## 1.5-approx.



$T_{Odd}$

Minimum  
Odd-vertex  
Matching

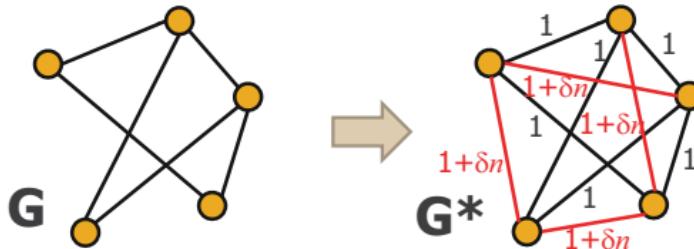
- กราฟ Euler มีเส้นเชื่อมจาก  $E_{MST}$  (ยาว  $L_{MST}$ ) และ  $E_{MOM}$  (ยาว  $L_{MOM}$ )
- ให้  $T_{Odd}$  คือ tour สั้นสุด (ยาว  $L_{Odd}$ ) ที่ผ่านเฉพาะปมใน  $V_{Odd}$ 
  - จะได้ว่า  $L_{Odd} \leq L_{Opt}$
- สามารถแบ่ง  $T_{Odd}$  ออกเป็นเส้นเชื่อมเส้นเว้นเส้นได้ 2 ชุด
  - แต่ละชุดเป็นการจับคู่ปมใน  $V_{Odd}$  ( $L_{MOM} \leq L_{\text{สัม}}, L_{\text{ต่ำ}}$ )
  - ชุดเล็กมีความยาวรวม  $\leq 0.5L_{Odd}$ , ดังนั้น  $L_{MOM} \leq 0.5L_{Odd}$
  - สรุปได้ว่า  $L_{MOM} \leq 0.5L_{Opt}$
- กราฟ Euler มีความยาวเส้นเชื่อมรวม  $L_{MST} + L_{MOM} \leq 1.5L_{Opt}$

## General TSP

- ❖ มี polynomial-time approx. algo.  
เพื่อแก้ปัญหา TSP ของกราฟใด ๆ หรือไม่ ?
- ❖ ถ้ามี จะแสดงให้เห็นว่า HAMILTONIANCYCLE  $\in \text{P}$ 
  - ❖ หมายความว่า สามารถพิสูจน์ได้ว่า  $P = NP$  !!!
  - ❖ แต่ยังไม่จริง แสดงว่า ยังไม่มีครอฟ polynomial-time approx. algo. สำหรับปัญหา TSP ของ  $G$  ทั่วไป

# ใช้ $\delta$ -approx TSP แก้ปัญหา HAMC

- ❖ ปัญหา :  $G=(V,E)$  มี Hamiltonian Cycle หรือไม่
- ❖ แปลง  $G$  ของ HAMILTONCYCLE เป็น  $G^*$  ของ TSP
  - ❖ สร้างกราฟ  $G^*=(V, E+E^*)$  เป็น complete graph
  - ❖ เส้นเชื่อมใน  $E$  ยาว 1, ใน  $E^*$  ยาว  $1+\delta n$  ( $n = |V|$ ,  $\delta > 1$ )
  - ❖ ถ้า  $G$  มีวงแหวนลิดัน,  $L_{Opt}$  ของ  $G^*$  ยาว  $n$
  - ❖ ถ้า  $G$  ไม่มีวงแหวนลิดัน,  $L_{Opt}$  ของ  $G^*$  ยาวเกิน  $\delta n$



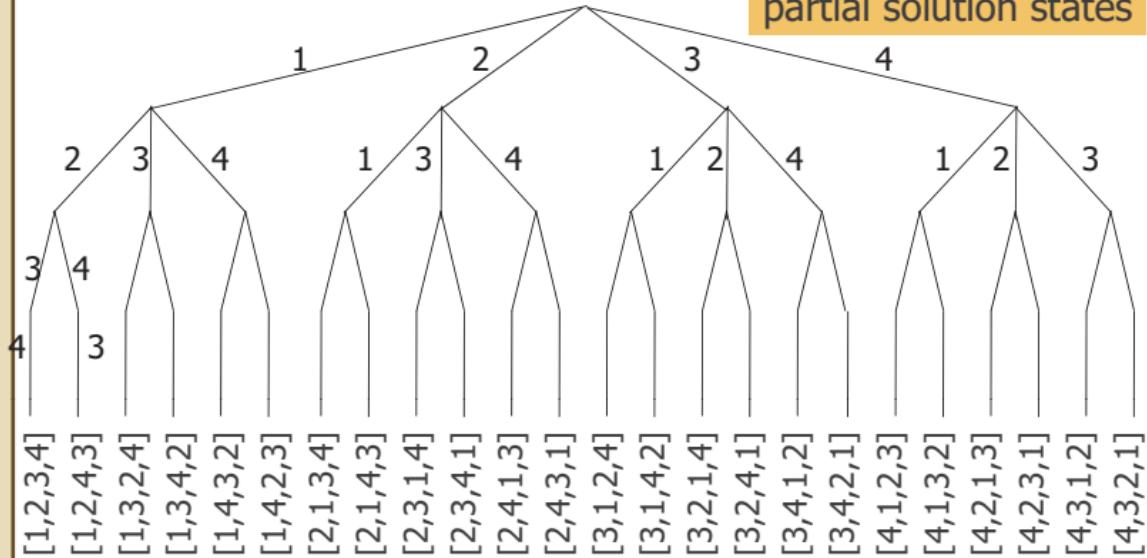
# ใช้ $\delta$ -approx TSP แก้ปัญหา HAMC

- ❖ ปัญหา :  $G=(V,E)$  มี Hamiltonian Cycle หรือไม่
- ❖ แปลง  $G$  ของ HAMILTONCYCLE เป็น  $G^*$  ของ TSP
  - ❖ สร้างกราฟ  $G^* = (V, E+E^*)$  เป็น complete graph
  - ❖ เส้นเชื่อมใน  $E$  ยาว 1, ใน  $E^*$  ยาว  $1+\delta n$  ( $n = |V|$ ,  $\delta > 1$ )
  - ❖ ถ้า  $G$  มีวงแย้มิลตัน,  $L_{Opt}$  ของ  $G^*$  ยาว  $n$
  - ❖ ถ้า  $G$  ไม่มีวงแย้มิลตัน,  $L_{Opt}$  ของ  $G^*$  ยาวเกิน  $\delta n$
- ❖ ถ้ามี approxTSP ที่เป็น  $\delta$ -approx
  - ❖ approxTSP( $G^*$ ), ต้องได้ tour ยาว  $\leq \delta \times$ ความยาวทางสั้นสุด
  - ❖ ถ้า  $G$  มีวงแย้มิลตัน, approxTSP( $G^*$ ) ให้ tour ยาว  $n$ 
    - ❖ เพราะถ้าไปผ่านเส้นที่ยาว  $1+\delta n$  จะได้ tour ยาวเกิน  $\delta n$
  - ❖ ถ้า  $G$  ไม่มีวงแย้มิลตัน, approxTSP( $G^*$ ) ให้ tour ยาวเกิน  $\delta n$
  - ❖ ถ้า approxTSP ใช้เวลา polynomial time, HAMC  $\in P$

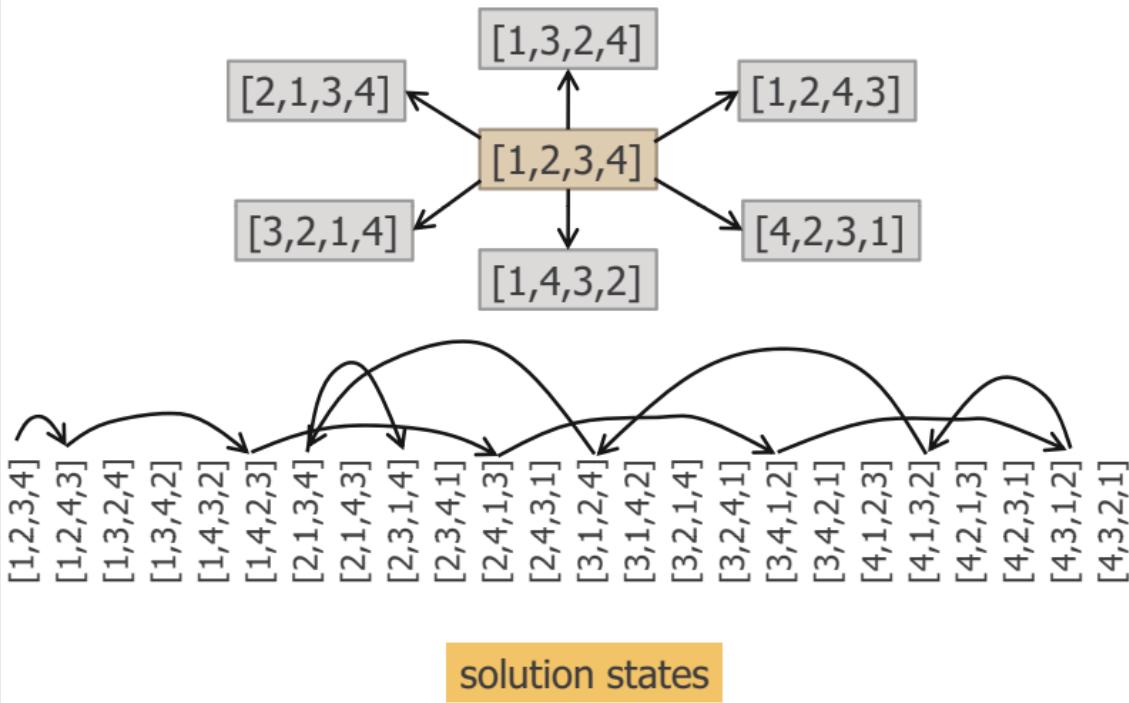


# Local Search

# การค้นแบบค่อยๆ สร้างผลเฉลย



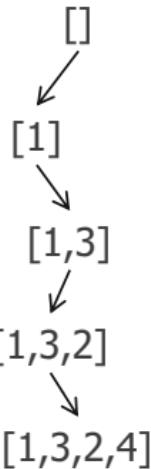
# การค้นเฉพาะที่ (Local Search)



# Local Search

- ❖ ใช้กับ NP-hard optimization problems
- ❖ "ไม่ได้นิยงคำตอบดีสุด ขอแค่คำตอบ "ดี ๆ"
- ❖ "ไม่สนใจ "เส้นทาง" การสร้างคำตอบ
- ❖ ค้นในปริภูมิผลเฉลย (solution space)
  - ❖ ทุกสถานะเป็นสถานะผลเฉลย
  - ❖ หยุดค้นเมื่อได้ ก็ได้คำตอบ

```
localSearch( p ) {  
    s = initialSolution(p)  
    while not terminate(p, s) {  
        s* = select( neighbor(p, s) )  
        if ( accept(p, s, s*) ) s = s*  
    }  
}
```



# Neighboring Solutions

❖ ปรับผลเฉลยหนึ่ง ให้เป็นผลเฉลยใหม่

❖ ตัวอย่าง

❖ TSP : [1,2,3,4]

❖ [1,**3**,**2**,4], [1,2,**4**,3], [1,**4**,3,2]

❖ VertexCover : [1,0,0]

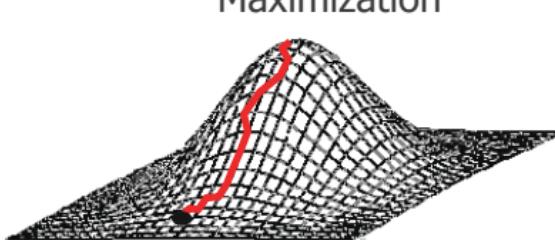
❖ [**0**,0,0], [1,**1**,0], [1,0,**1**]

❖ 3-Coloring : [1,2,2]

❖ [**2**,2,2], [**3**,2,2], [1,**1**,2], [1,**3**,2], [1,2,**1**], [1,2,**3**]

# Hill Climbing

```
knapsack( w[1..n], v[1..n], W ) {  
    maxV = 0  
    maxX[1..n] = a new array[1..n]  
    betterSolution = true  
    while (betterSolution) {  
        x = maxX  
        betterSolution = false  
        for (i=1; i<n; i++) {  
            x[i] = 1 - x[i]  
            Σv = sum(v, x)  
            if (sum(w, x) <= W && Σv > maxV) {  
                maxX = x; maxV = Σv; betterSolution = true  
                break  
            }  
            x[i] = 1 - x[i]  
        }  
    }  
    return maxX  
}
```



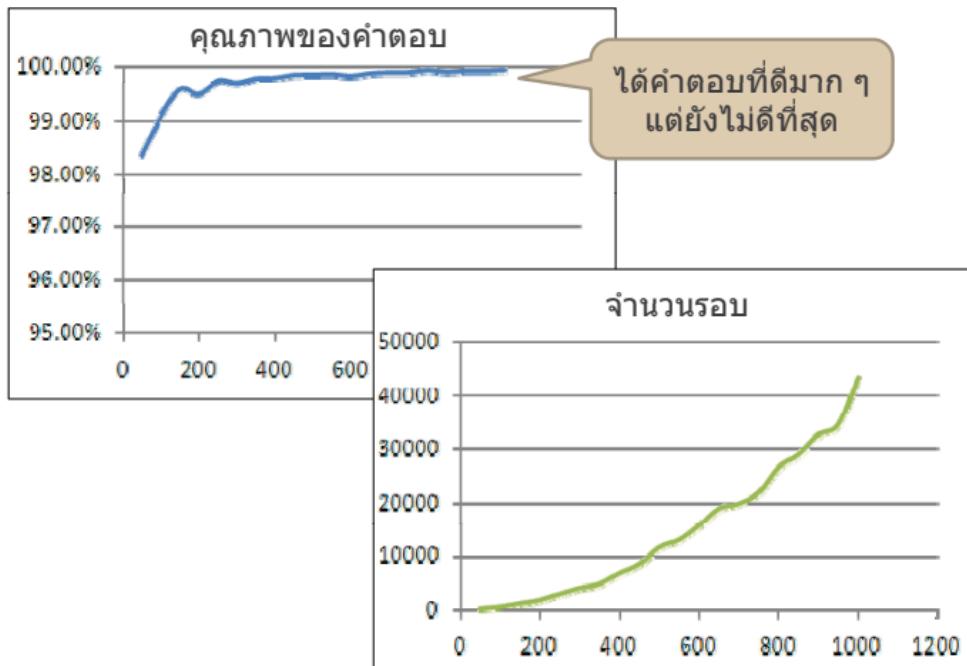
Maximization

Simple Ascent  
vs.  
Steepest Ascent

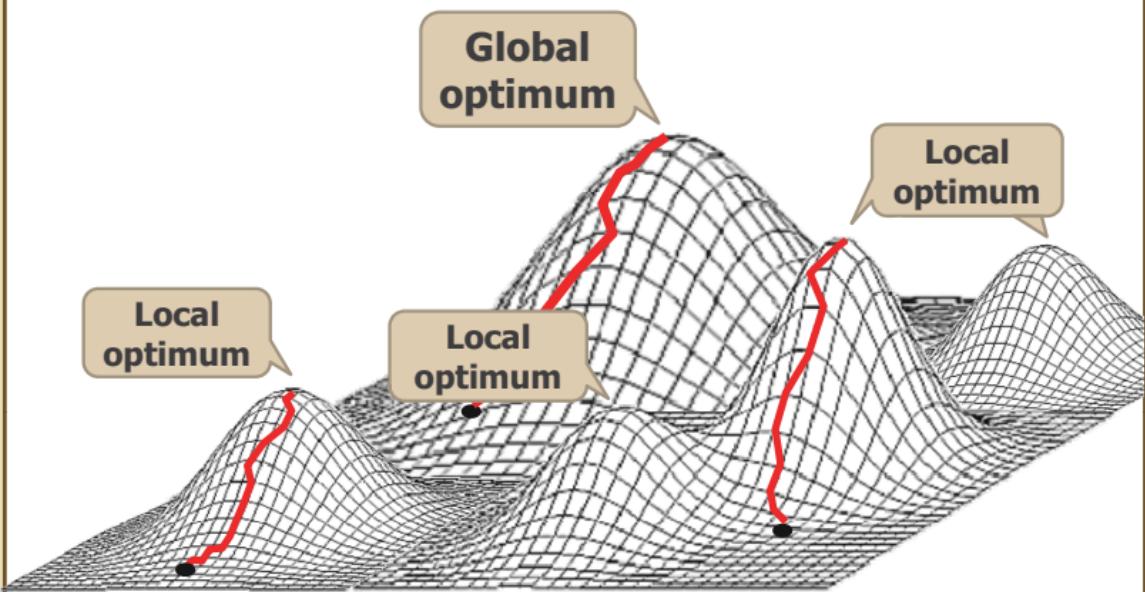
สูงกว่าขึ้นเลย

ขึ้นไปทางที่ชันสุด

# Hill Climbing : 0/1 Knapsack



# Global กับ Local Optimum



# Hill Climbing : TSP (ปืนลง)

```
tsp( d[1..n] [1..n] ) {
    minT = {1,2,3,...,n}; shuffle( minT )
    minL = tourLength(d, minT);
    betterSolution = true;
    while ( betterSolution ) {
        x = minT; betterSolution = false;
        for each pair (i,j) in (1..n, 1..n) {
            x[i] ↔ x[j]
            len = tourLength(d, x)
            if (len < minL) {
                minL = len; minT = x; betterSolution = true;
                break;
            }
            x[i] ↔ x[j]
        }
    }
    return minT;
}
```

[1,2,3,4]

[1,3,2,4]

[1,2,4,3]

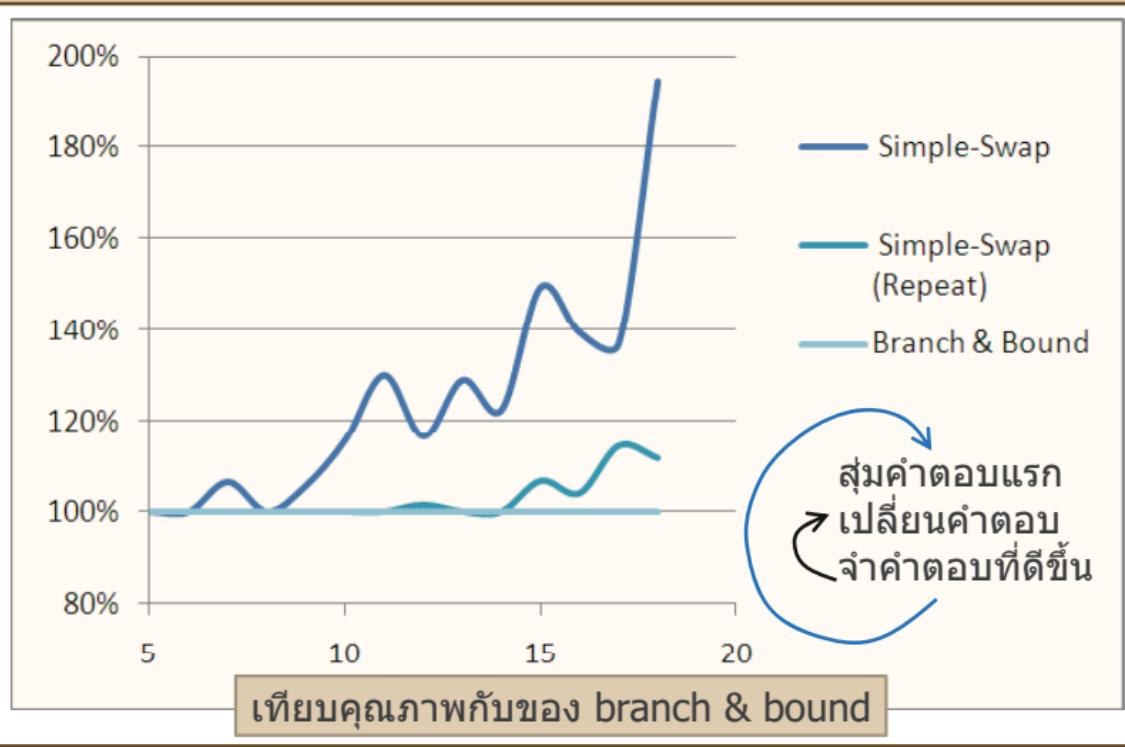
[1,4,3,2]

ต่างกว่าลงเลย

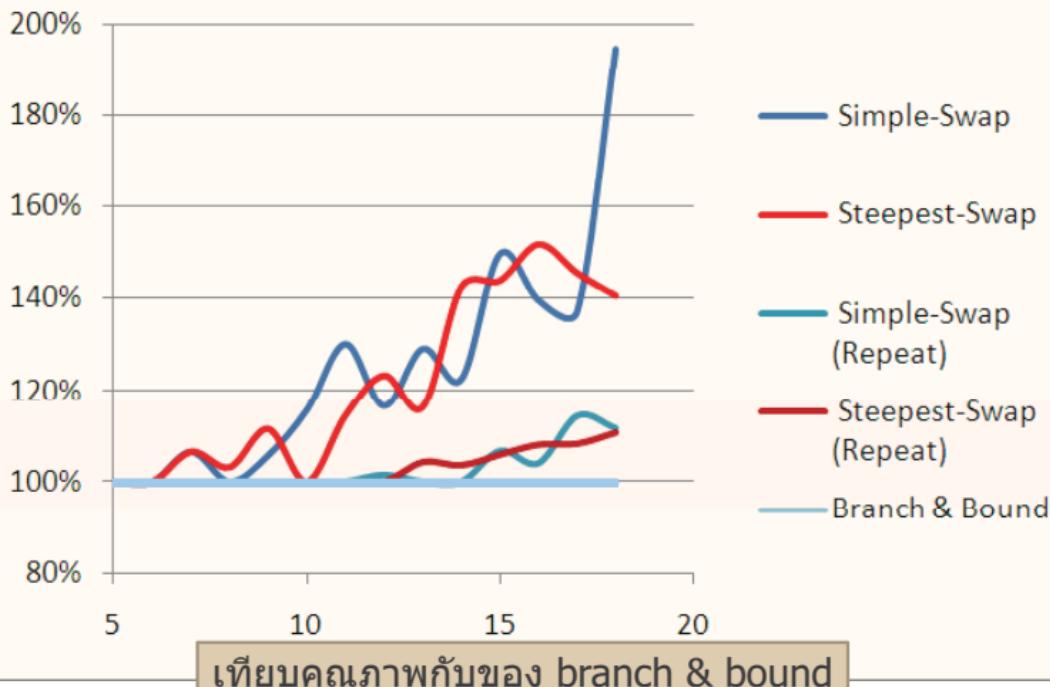
Simple Descent  
vs.  
Steepest Descent

ลงไปทางที่ชันสุด

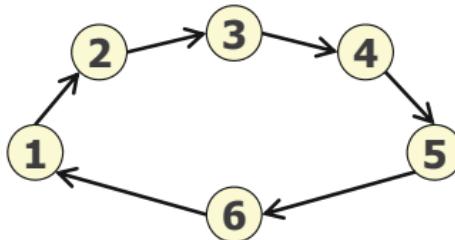
# TSP : Simple Descent



# Simple กับ Steepest Descent



# Neighboring Functions



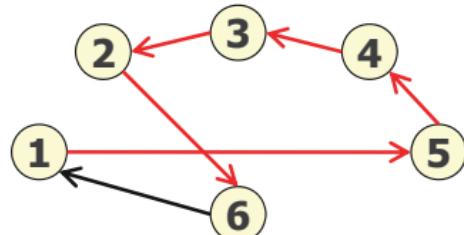
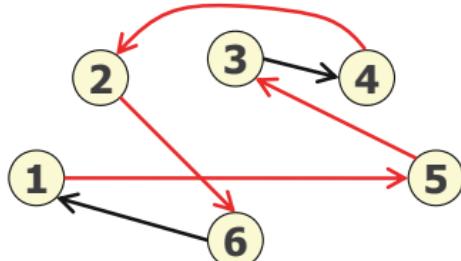
swap

1,5,3,4,2,6

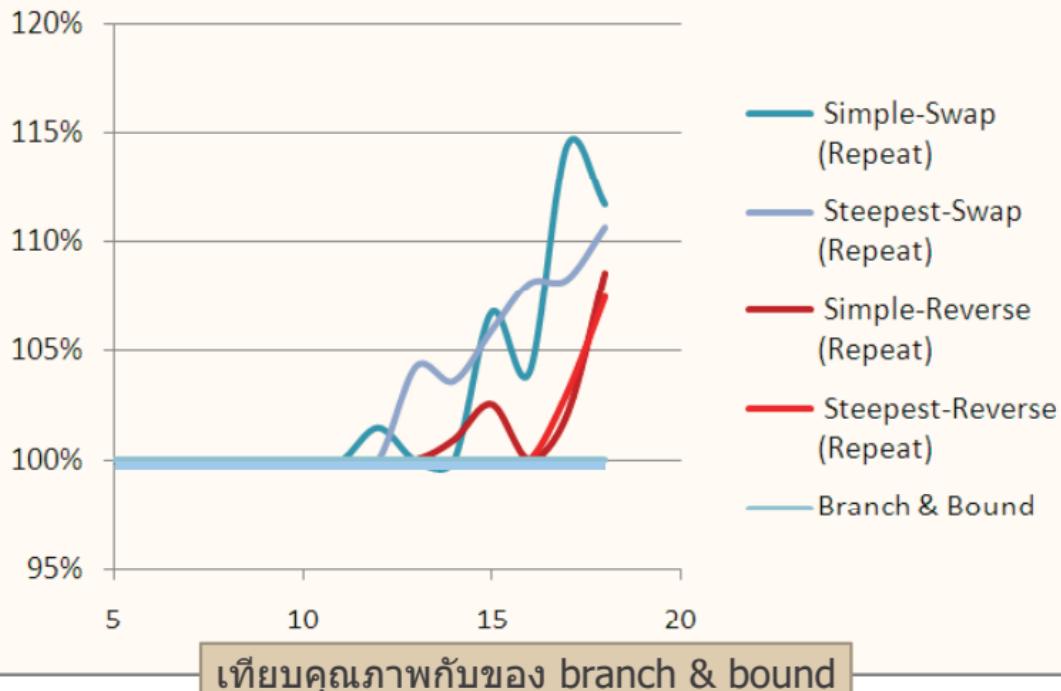
1,2,3,4,5,6

reverse

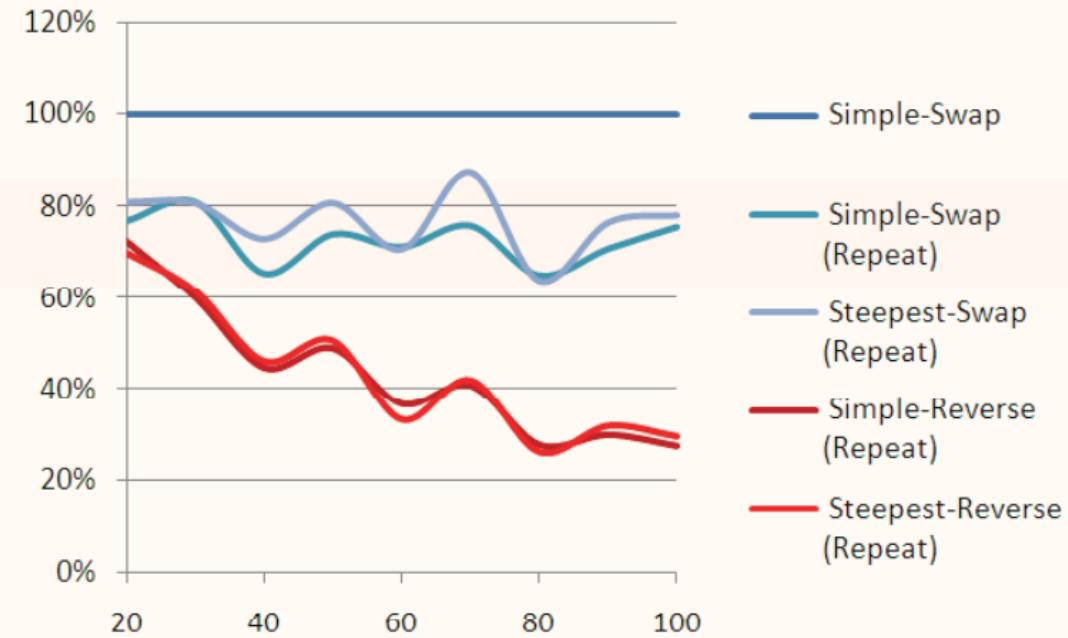
1,5,4,3,2,6



# Neighboring Functions



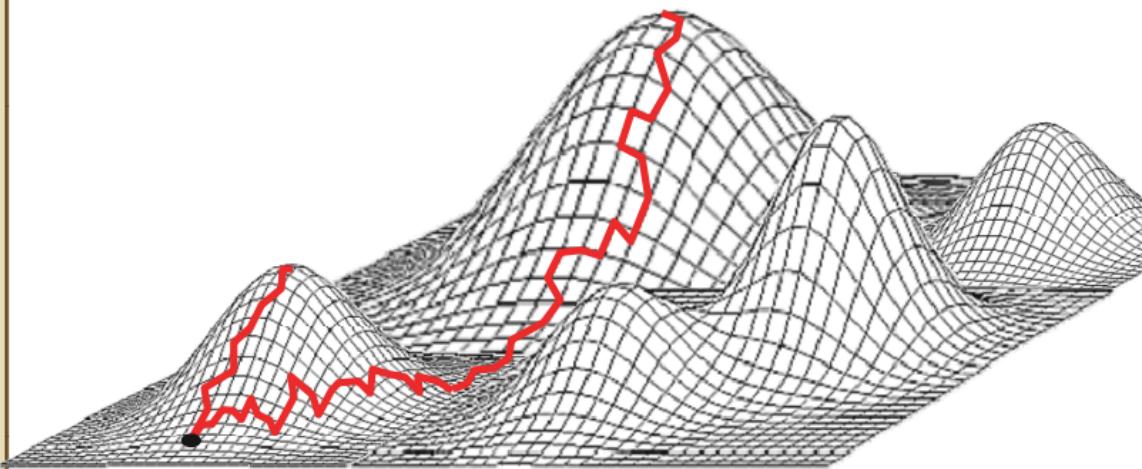
# Neighboring Function



เทียบคุณภาพกับของ Simple-Swap

# ทำอย่างไรให้ไม่ติด local optimum

- ❖ ต้องยอมรับคำตอบใหม่ แม้จะมีคุณภาพต่ำกว่า

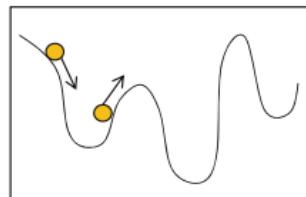


# Simulated Annealing

## ❖ ล้อเลียนกระบวนการอบเนื้อยา (Annealing)

- ❖ เพิ่มอุณหภูมิให้ของแข็งหลอมละลายกลายเป็นของเหลว แล้วลดอุณหภูมิลงให้กลับสภาพเป็นของแข็ง
- ❖ อบเนื้อยาโลหะ
  - ❖ ลดอุณหภูมire็วไป → เปราะ x
  - ❖ ค่อย ๆ ลดอุณหภูมิ → เนื้อยา ✓
- ❖ โครงสร้างเปลี่ยนแปลงจาก สภาวะพลังงานศักย์สูง → สภาวะพลังงานศักย์ต่ำ
- ❖ อาจเปลี่ยนจากต่ำ → สูงได้
  - ❖ อุณหภูมิสูงเปลี่ยนมาก ต่ำเปลี่ยนน้อย

$$e^{-\Delta E / KT}$$



# Simulated Annealing

```

localSearch( p ) {
    s = initialSolution(p)
    while not terminate(p, s) {
        s* = select( neighbor(p, s) )
        if ( accept(p, s, s*) ) s = s*
    }
}

simulatedAnnealing( p ) {
    s = initialSolution(p)
    T = T0
    while ( T > 0 ) {
        for (i = 1; i <= N; i++) {
            s* = neighbor(p, s)
            ΔE = energy(s*) - energy(s)
            if (ΔE < 0 OR random(0,1) < e-ΔE/kT )
                s = s*
        }
        T = αT
    }
}

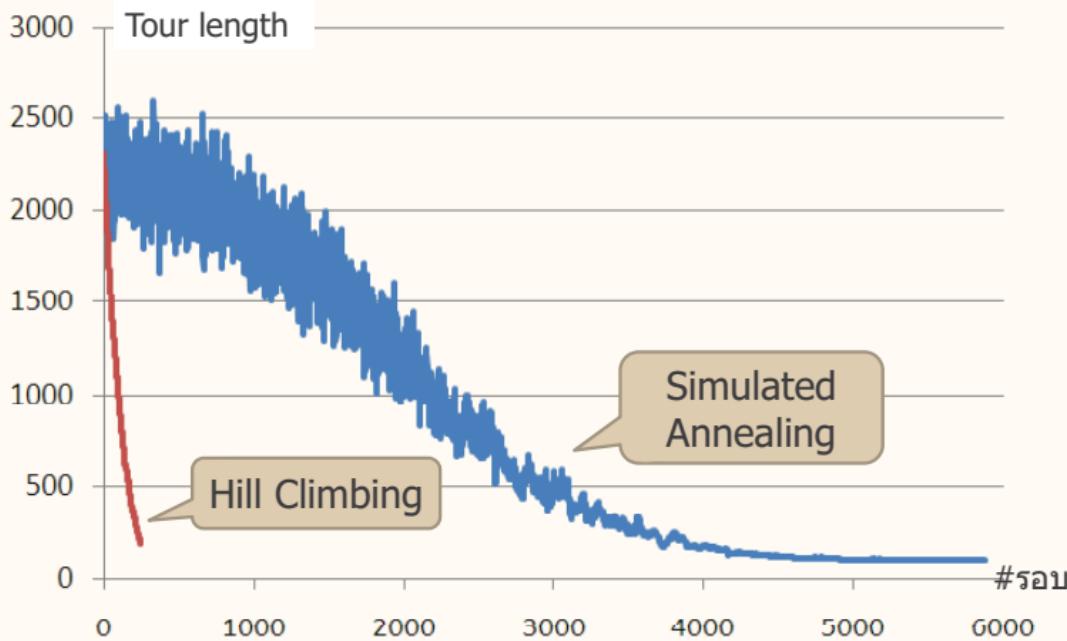
```

# TSP : Simulated Annealing

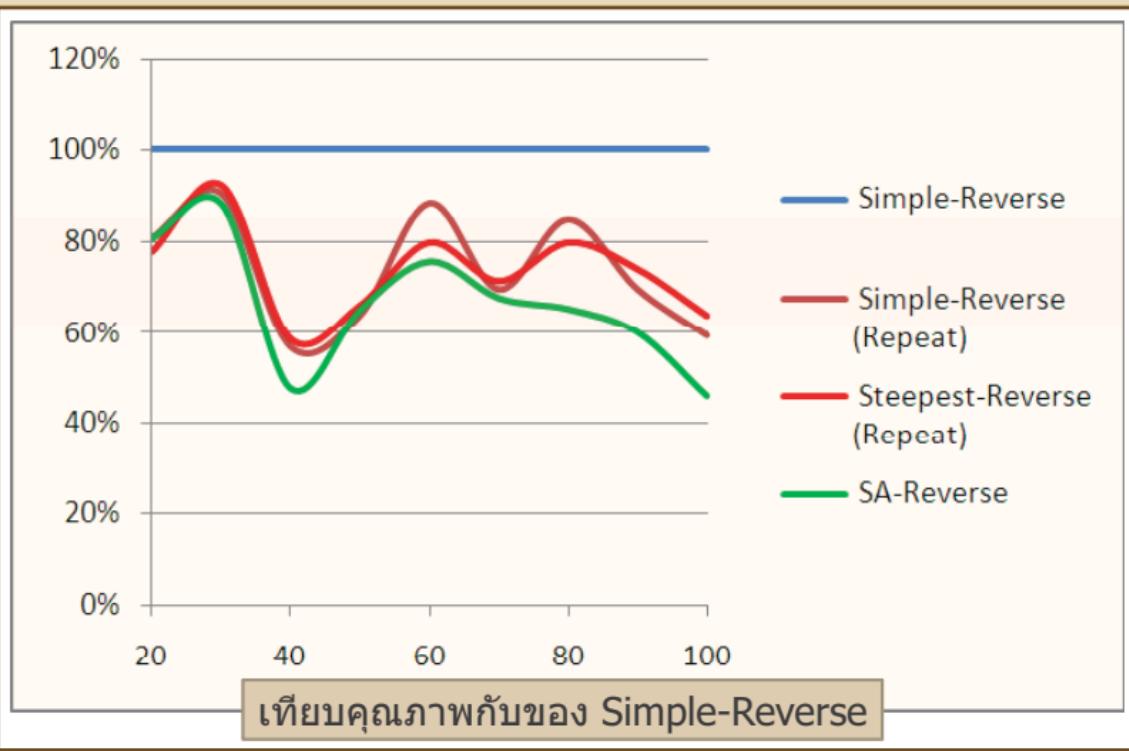
```
tspSA( d[1..n][1..n] ) {
    tour = initialTour( d )
    maxT = T = tourLength(d, tour)
    while ( T > 0.001 ) {
        N = maxT - T;
        for (i = 1; i <= N; i++) {
            newTour = nextTour( minT )
            dE = tourLength(d,newTour) - tourLength(d,tour)
            if (dE < 0 OR random(0,1) < e-dE/kT ) {
                tour = newTour;
            }
        }
        T *= 0.999
    }
    return tour;
}
```

รับเส้นทางที่ยาวกว่าได้ง่ายเมื่อ T สูง  
รับยกขั้นเมื่อ T ต่ำลง

# Hill Climb.. กับ Simulated Annealing...



# TSP : Simulated Annealing

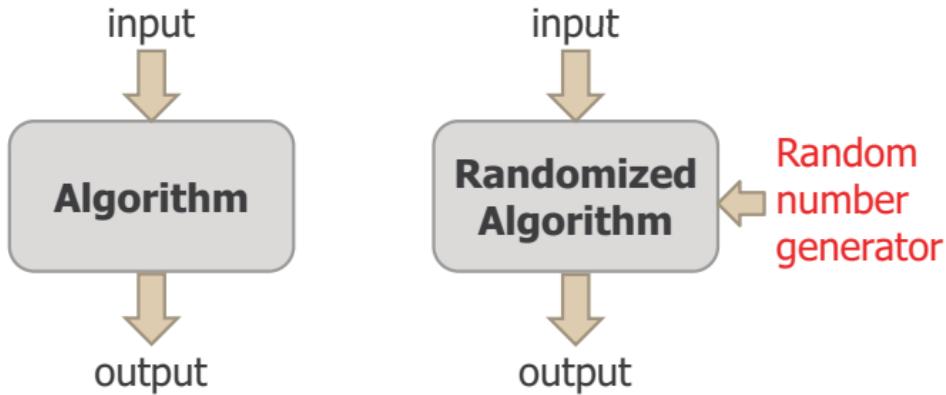


# อัลกอริทึมเชิงสุ่ม

สมชาย ประสิทธิ์จุตระกุล

ผู้สอน กานต์ ภู่ว่องไว

# อัลกอริทึมเชิงสุ่ม



ง่าย, เร็ว, หรือ ง่าย+เร็ว  
ไม่กล้าพนัง ให้โยนเหรียญ  
จนปัญญา !

# Randomized Majority

- ❖ ถ้า  $d$  ไม่มีตัวหมุ่มาก  $2,1,3,1,0,2,1,4,5,7,8,3,2,8$ 
  - ❖ สุ่มตัวใดก็ต้องไม่ใช่ คำตอบถูกเสมอ
- ❖ ถ้า  $d$  มีตัวหมุ่มาก  $1,2,1,1,1,1,2,1,1,1,1,2,3,1$ 
  - ❖ โอกาสสุ่มได้ตัวหมุ่มาก  $> 1/2$  โอกาสสตอปผิด  $< 1/2$

```

hasMajority( d[1..n] ) {
    i = random(1, n)
    c = count( d, i )
    return (c > n/2)           โอกาสผิด < 1/2
}
hasMajorityRepeat( d[1..n], k ) {
    for (i=1; i<=k; i++) {
        if (hasMajority(d)) return true
    }
    return false                โอกาสผิด < 1/2k
}

```

# Randomized Quicksort

## ❖ เวลาของ Quicksort ขึ้นกับ pivot ที่เลือกดอนเบ่ง

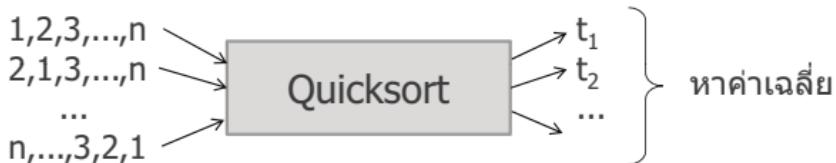
- ❖ ถ้าเลือก pivot แบบตามตัว จะมี bad input
  - ❖ worst-case เป็น  $O(n^2)$
  - ❖ avg-case เป็น  $O(n \log n)$  (เฉลี่ย input ทุกรูปแบบ)
- ❖ ถ้าเลือก pivot แบบสุ่ม ไม่มี bad input
  - ❖ ได้ expected running time เป็น  $O(n \log n)$
  - ❖ ยังคงมี worst-case เป็น  $O(n^2)$  แต่ด้วยโอกาสที่ต่ำมาก

```
partition( d[1..n], left, right ) {  
    k = random(left, right)  
    d[k] ↔ d[left]  
    p = d[left] // pivot  
    i = left, j = right + 1  
    ...
```

# Average กับ Expected

## ❖ วิเคราะห์ average time

- ❖ เวลาการทำงานเฉลี่ยของ input ทุกรูปแบบ



## ❖ วิเคราะห์ expected time

- ❖ เวลาการทำงานเฉลี่ยของการสุ่มทุกรูปแบบ



# Randomized Algorithms

## ❖ Monte Carlo

- ❖ คืนค่าตอบเสมอ แต่อาจผิด (ด้วยความน่าจะเป็นที่ต่ำ)
- ❖ ทำงานรวดเร็ว
- ❖ ตัวอย่าง : Randomized Majority

## ❖ Las-Vegas

- ❖ คืนค่าตอบที่ถูกต้องเสมอ หรือไม่ก็ยอมรับว่าไม่มีคำตอบ
- ❖ อาจทำงานช้า (ด้วยความน่าจะเป็นที่ต่ำ)
- ❖ ตัวอย่าง : Randomized Quicksort

Metropolis,  
Ulam,  
Neumann,  
1949



Babai  
1979

# Monte Carlo Algorithms

- ❖ **p-correct** :  $\Pr[\text{ให้คำตอบถูกต้อง}] \geq p$ 
  - ❖ hasMajority เป็น  $1/2$  - correct
- ❖ **ถ้าเป็น decision problem**
  - ❖ One-sided error
    - ❖ ถ้าคืนคำตอบหนึ่ง จะมั่นใจแน่นอนว่าถูกต้อง
    - ❖ hasMajority เป็น one-sided error
      - คืน true (มีตัวหมุน) เป็นคำตอบที่ถูกต้อง
      - คืน false (ไม่มีตัวหมุน) เป็นคำตอบที่มีโอกาสผิด
  - ❖ Two-sided error : คำตอบที่คืน มีโอกาสผิด

# Monte Carlo : Amplification

- ❖ เพิ่มโอกาสที่ได้คำตอบถูกต้องด้วยการทำซ้ำ
  - ❖ ถ้าเป็น  $p$ -correct, one-sided error algorithm
  - ❖ เรียกซ้ำ  $k$  ครั้ง จะได้  $1 - (1 - p)^k$  - correct

ครั้งที่	โอกาสผิด	โอกาสถูก
1	$(1 - p)$	$p$
2	$(1 - p)^2$	$1 - (1 - p)^2$
$k$	$(1 - p)^k$	$1 - (1 - p)^k$

```
hasMajorityRepeat( d[1..n] , k ) {  
    for (i = 1 to k) {  
        if ( hasMajority(d) ) return true  
    }  
    return false  
}
```

# $C = A \times B ?$

- ❖ ให้  $A$ ,  $B$ , และ  $C$  เป็นเมทริกซ์ขนาด  $n \times n$
- ❖ อายากรทราบว่า  $C$  เท่ากับ  $A \times B$  หรือไม่
- ❖ วิธีทำ
  - ❖ หา  $A \times B$  และเปรียบเทียบกับ  $C$ 
    - ❖ คุณธรรมดा  $\Theta(n^3)$  เปรียบเทียบอีก  $O(n^2)$
    - ❖ ใช้การคูณของ Coppersmith & Winograd  $O(n^{2.38})$
    - ❖ ใช้ randomized algo.  $\rightarrow O(n^2)$

## **C = A×B ?**

- ❖ ให้  $D = C - A \times B$
- ❖ ให้  $X$  เป็น  $0/1$  เวกเตอร์ขนาด  $1 \times n$
- ❖ จะได้  $XD$  เป็นเวกเตอร์ขนาด  $1 \times n$
- ❖ ถ้า  $D = 0$  จะได้  $XD = 0$  แน่ (ไม่ว่า  $X$  จะมีค่าใด)
- ❖ ถ้า  $D \neq 0$   $XD$  อาจเท่าหรือไม่เท่ากับ  $0$  ก็ได้

$$\begin{array}{ccccc} 0 & 0 & 1 & 1 & 1 \\ X_1 & & & & \\ & 0 & 0 & 0 & 0 & 0 \\ & 0 & 0 & 0 & 0 & 0 \\ & 1 & 0 & 0 & 2 & -1 \\ & 0 & 0 & 0 & 0 & 2 \\ 1 & 0 & 1 & 0 & 1 & \\ & 1 & 0 & 0 & -2 & 1 \\ X_2 & & & & & \\ D & & & & & \\ & 0 & 0 & 0 & 0 & 0 \\ & 0 & 0 & 0 & 0 & 0 \\ & 0 & 0 & 0 & 0 & 0 \\ & 0 & 0 & 0 & 0 & 0 \end{array}$$
$$X_1 D$$
$$X_2 D$$

**C = A × B ?**

**X**  $x_1 \dots x_k \dots x_n$

ให้แก้วที่  $k$  มี  $d_{k,j} \neq 0$

**D**

$d_{1,1}$	$x_1$	$d_{1,j} \dots d_{1,n}$
$d_{2,1}$	$x_2$	$d_{2,j} \dots d_{2,n}$
.	.	.
$d_{k,1}$	$x_k$	$d_{k,j} \dots d_{k,n}$
.	.	.
$d_{n,1}$	$x_n$	$d_{n,j} \dots d_{n,n}$

$$x_1d_{1,j} + x_2d_{2,j} + \dots + x_k d_{k,j} + \dots + x_n d_{n,j} \quad \Pr[\neq 0] \geq 1/2$$

$$x_1d_{1,j} + x_2d_{2,j} + \dots + 0 d_{k,j} + \dots + x_n d_{n,j} \quad \cancel{\neq 0}$$

$$x_1d_{1,j} + x_2d_{2,j} + \dots + 1 d_{k,j} + \dots + x_n d_{n,j} \quad \cancel{\neq 0}$$

ถ้า  $d_{k,j} \neq 0$  และสุ่มให้  $x_k = 0$  หรือ  $1$  ,  $\Pr[ \mathbf{X}\mathbf{D} \neq 0 ] \geq 1/2$

ถ้า  $\mathbf{D} \neq \mathbf{0}$  สุ่มให้  $\mathbf{x}[1..n] = 0$  หรือ  $1$  ,  $\Pr[ \mathbf{X}\mathbf{D} \neq 0 ] \geq 1/2$

# **C = A×B ?**

ถ้า  $D \neq 0$  สุ่มให้  $X[1..n] = 0$  หรือ  $1$  ,  $\Pr[XD = 0] < 1/2$

```

equalsMC( A[1..n][1..n], B[1..n][1..n],
           C[1..n][1..n] ) {
    D = C - Ax B
    for( i = 1 to n ) X[i] = random(0,1)
    return XD == 0
}

```

คืน false, มั่นใจแน่ว่าไม่เท่า

one-sided error

คืน true, มั่นใจ  $\geq 50\%$  ว่าเท่า

$1/2$  - correct

เราไม่คำนวณ  $XD = X(C - A \times B)$  คำนวณ  $A \times B$  ใช้  $O(n^{2.38})$   
แต่จะคำนวณ  $XC$  กับ  $(XA)B$  และเปรียบเทียบ  $XC == (XA)B$

$\Theta(n^2)$        $\Theta(n^2)$        $\Theta(n^2)$        $\Theta(n^2)$

# C = A×B ?

```

equalsMC( A[1..n][1..n], B[1..n][1..n],
           C[1..n][1..n] ) {
    for( i = 1 to n ) X[i] = random(0,1)
    return XC == (XA)B
}

```

1/2 - correct

```

equalsMC_Repeat( A[1..n][1..n], B[1..n][1..n],
                  C[1..n][1..n], k ) {
    for (i = 1 to k) {
        if ( NOT equalsMC(A, B, C) ) return false
    }
    return true
}

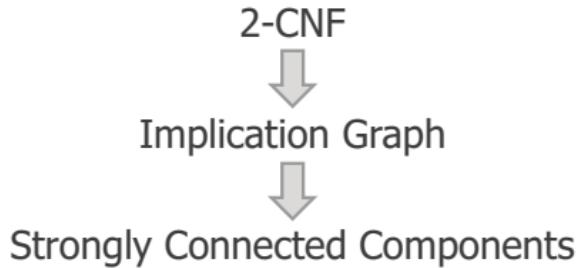
```

(1 - 2<sup>-k</sup>) - correct

# 2-SAT

$$(x_1 + \bar{x}_2)(x_1 + x_3)(\bar{x}_1 + x_2)(\bar{x}_2 + \bar{x}_3)(\bar{x}_2 + \bar{x}_4)(x_1 + \bar{x}_4)$$

2-SAT ∈ P



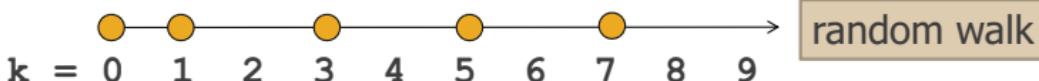
# **2-SAT\_MC**

- ❖ สุ่ม truth assignment,  $x[i] \leftarrow \text{random}(0,1)$
  - ❖ ถ้ายังมี clause ที่เป็นเท็จ
    - ❖ เลือกมาหนึ่ง แล้วสุ่มเลือกหนึ่งในสอง literal ของ clause นั้น
    - ❖ กลับค่าของ literal นั้น

$x_1, x_2, x_3, x_4$	$(x_1 + \bar{x}_2)(x_1 + x_3)(\bar{x}_1 + x_2)(\bar{x}_2 + \bar{x}_3)(\bar{x}_2 + \bar{x}_4)(x_1 + \bar{x}_4)$
$\begin{pmatrix} 0 \\ 1 \end{pmatrix} \quad 1 \quad 1 \quad 0$	$(\begin{pmatrix} 0 \\ 1 \end{pmatrix} \quad 0)(0 \quad 1)(1 \quad 1)(\begin{pmatrix} 0 \\ 1 \end{pmatrix} \quad 0)(0 \quad 1)(0 \quad 1)$
$\begin{pmatrix} 1 \\ 1 \end{pmatrix} \quad 1 \quad 0$	$(\begin{pmatrix} 1 \\ 1 \end{pmatrix} \quad 0)(1 \quad 1)(0 \quad 1)(\begin{pmatrix} 0 \\ 1 \end{pmatrix} \quad 0)(0 \quad 1)(1 \quad 1)$
$\begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad 1 \quad 0$	$(1 \quad 1)(1 \quad 1)(\begin{pmatrix} 0 \\ 1 \end{pmatrix} \quad 0)(1 \quad 0)(1 \quad 1)(1 \quad 1)$
$\begin{pmatrix} 0 \\ 0 \end{pmatrix} \quad 1 \quad 0$	$(0 \quad 1)(0 \quad 1)(\begin{pmatrix} 1 \\ 1 \end{pmatrix} \quad 0)(1 \quad 0)(1 \quad 1)(1 \quad 1)$

## 2-SAT\_MC

- ❖ มีตัวแปร  $n$  ตัว,  $x_1, x_2, \dots, x_n$
- ❖ ถ้ามี **satisfying truth assignment**
  - ❖ ให้  $k$  คือจำนวนของ  $x$  ที่มีค่าถูกต้อง
    - ❖ ถ้า  $k = 0$ , รอบหน้า  $k = 1$
    - ❖ ถ้า  $k \neq 0$ , รอบหน้า  $k$  มีค่าเพิ่มหนึ่ง หรือไม่ลดหนึ่ง



- ❖ ให้  $s(k)$  คือจำนวนก้าวที่ต้องเดินเมื่อยุ่งที่ตำแหน่ง  $k$ 
  - ❖  $s(n) = 0, s(0) = 1 + s(1)$
  - ❖  $s(k) \leq 1 + \frac{1}{2}(s(k+1) + s(k - 1))$
  - ❖ แก้ recurrence นี้ จะได้  $s(0) = n^2$
  - ❖ ค่าคาดหวังของจำนวนก้าวที่ต้องเดินคือ  $n^2$

# 2-SAT\_MC

- ❖ ถ้า 2-SAT\_MC คืน true  $\rightarrow$  ถูกต้องแน่
- ❖ ถ้า 2-SAT\_MC คืน false
  - ❖ random walk จนครบแล้ว ไม่พบ satisfying assignment
    - ❖ เพราะว่า unsatisfiable จริง ๆ
    - ❖ หรือ satisfiable แต่ยังเดินไม่ถึง
- ❖ ถ้า satisfiable และให้เดิน  $2n^2$  ครั้ง
  - ❖ จาก Markov's inequality :  $\Pr[X \geq a] \leq E(X) / a$
  - ❖  $\Pr[\text{จำนวนก้าวที่ต้องเดิน} \geq 2n^2] \leq n^2 / 2n^2 = 1/2$
- ❖ 2-SAT\_MC เป็น  $1/2$  - correct เมื่อให้เดิน  $2n^2$  ก้าว
- ❖ ขยายความแม่นยำโดยทำซ้ำ k ครั้ง
  - ❖  $(1 - 2^{-k})$  - correct

# Primality Testing

- ❖ PRIMES  $\in \mathbb{P}$  AKS = Agrawal-Kayal-Saxena
- ❖ 2002 : AKS Primality Test  $\rightarrow O(\log^{12+\epsilon}(n))$
- ❖ ได้รับการปรับปรุง  $\rightarrow O(\log^{6+\epsilon}(n))$
- ❖ Miller-Rabin test  $\rightarrow O(\log^3(n))$ 
  - ❖  $\frac{3}{4}$  - correct, one-sided error

# Fermat Test

ถ้า  $n$  เป็นจำนวนเฉพาะ  $a^{n-1} \bmod n = 1$  สำหรับ  $1 \leq a \leq n - 1$

$$2^{3-1} \bmod 3 = 1, \quad 4^{7-1} \bmod 7 = 1 \quad \rightarrow \text{prime}$$

$$2^{9-1} \bmod 9 = 4, \quad 2^{25-1} \bmod 25 = 16 \quad \rightarrow \text{composite}$$

$$4^{15-1} \bmod 15 = 1 \text{ แต่ } 15 = 3 \times 5 \quad \rightarrow \text{false witness}$$

จำนวน	จำนวน false witness
561	381
1105	766
1729	1294

มีจำนวนมากมายที่มี false witness มาก

ไม่เป็น p-correct

# Miller - Rabin

ถ้า  $n$  เป็นจำนวนเฉพาะ,  $a^{n-1} \bmod n = 1$  สำหรับ  $1 \leq a \leq n-1$

$$a^k \bmod n = \begin{cases} (a^{\lfloor k/2 \rfloor} \bmod n)^2 \bmod n & k \text{ is even} \\ a(a^{\lfloor k/2 \rfloor} \bmod n)^2 \bmod n & k \text{ is odd} \end{cases}$$

```
MillerRabinTest( n ) { // Is n prime ?  
    a = random(2, n-2) // ไม่เลือก 1 กับ n-1  
    return powerMod(a, n-1, n) == 1  
}  
  
powerMod(a, k, n) {  
    if (k == 0) return 1  
    d = powerMod(a, k/2, n)  
    x = d2 % n  
    if (x == 1 AND d != 1 AND d != n-1) return 0  
    if (k % 2 == 1) d = x * a % n  
    return d;  
}
```

มี false witness  
ไม่เกิน  $(n-9)/4$  ตัว

3/4 - correct

ถ้า  $n$  เป็นจำนวนเฉพาะ และ  $1 \leq d \leq n-1$ ,  
 $d^2 \bmod n = 1$  ก็ต่อเมื่อ  $d=1$  กับ  $d=n-1$

# Amplification : ลดโอกาสผิด

```
isProbablePrime( n, k ) { // Is n prime ?  
    for (i = 1 to k) {  
        if (MillerRabinTest(n) ≠ 1) return false  
    }  
    return true;  
}
```

โอกาสผิด  $(\frac{1}{4})^k$

$(1 - (\frac{1}{4})^k)$  - correct

# Las Vegas

- ❖ คืนคำตอบที่ถูกต้องเสมอ
  - ❖ หรือไม่ก็ยอมรับว่าไม่มีคำตอบ
  - ❖ เมื่อไม่มีคำตอบ ก็ลองใหม่
- ❖ ใช้การสุ่มเพื่อตัดสินใจระหว่างการทำงาน
- ❖ อาจทำงานช้า (ด้วยความน่าจะเป็นที่ต่ำ)
- ❖ ตัวอย่าง :
  - ❖ Randomized Quicksort
  - ❖ Randomized Quickselect
  - ❖ ...

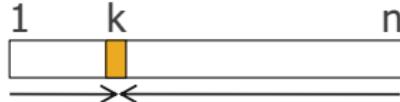
# Randomized Sequential Search

## ❖ ค้นตามลำดับจากซ้ายไปขวาในรายการ (กรณีพบร)

- ❖ ช้าสุด (พบตัวท้ายสุดในรายการ) เปรียบเทียบ  $n$  ครั้ง
- ❖ เฉลี่ย (ข้อมูลทุกตัวในรายการ) เปรียบเทียบ  $(n+1)/2$  ครั้ง

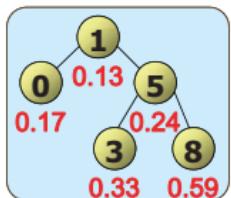
## ❖ Las Vegas $(1+2+3+\dots+n)/n = (n(n+1)/2) / n$

- ❖ โยนเหรียญ
  - ❖ ออกรหัส : ค้นจากซ้ายไปขวา
  - ❖ ออกรกอย : ค้นจากขวามาซ้าย
- ❖ ต้องการค้นตัวที่  $k$  ในรายการ
  - ❖ ออกรหัส : เปรียบเทียบ  $k$  ครั้ง  $k \quad (n - k + 1)$
  - ❖ ออกรกอย : เปรียบเทียบ  $(n - k + 1)$  ครั้ง
- ❖ expected #cmps  $= \frac{1}{2}k + \frac{1}{2}(n - k + 1)$   
 $= (n+1)/2$  ("ไม่ว่าจะค้นตัวใดก็ตาม)



# ต้นไม้ทรีพ (Treap)

- ❖ นำแนวคิดของสีปเพิ่มในต้นไม้คันหาแบบทวิภาค
- ❖ แต่ละปมมีข้อมูลสองตัว
  - ❖ ตัวข้อมูลจริง ของต้นไม้คันหาแบบทวิภาค
  - ❖ ข้อมูลเสริม เมื่อมองต้นไม้นี้เป็นสีปแบบน้อยสุด
- ❖ ต้องรักษาโครงสร้างของต้นไม้ให้คงคุณสมบัติ
  - ❖ ข้อมูลจริงของต้นไม้ซ้ายน้อยกว่าขวา ของต้นขามากกว่าขวา
  - ❖ ข้อมูลเสริมของปมพ่อต้องน้อยกว่าของลูก
  - ❖ ใช้การหมุนปม สลับความเป็นพ่อลูกได้

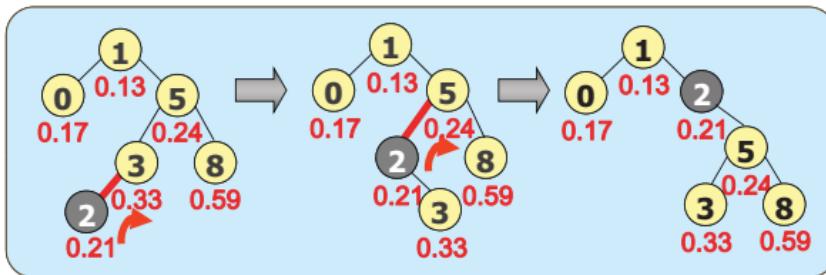


ข้อมูลเสริมของสีปฯได้มาจากการสุ่มจำนวนจริงตอนสร้างปม

# การเพิ่มข้อมูล

## ❖ ต้องการเพิ่ม x

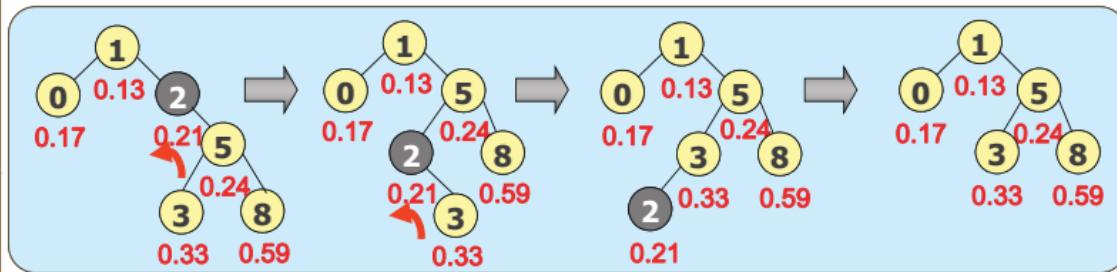
- ❖ เพิ่มปมใหม่เก็บ x เหมือนการเพิ่มในต้นไม้ค้นหาแบบทวิภาค
- ❖ สูนจำนวนจริงเป็นข้อมูลเสริมกำกับปมใหม่
- ❖ หมุนปม x ขึ้นไปเรื่อย ๆ จนกว่าจะไม่ผิดอันดับแบบฮีป



# การลบข้อมูล

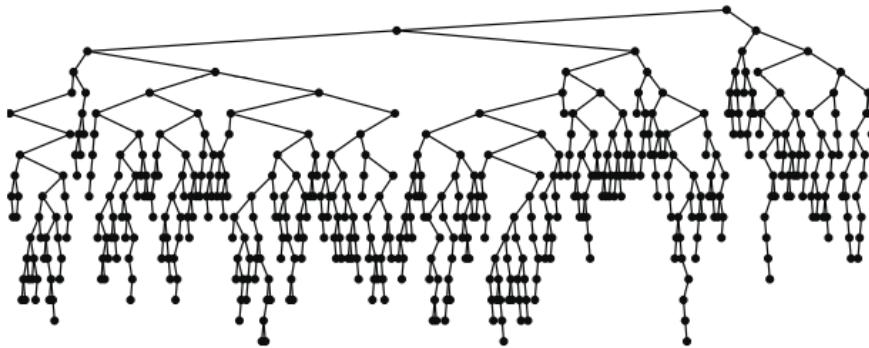
## ❖ ต้องการลบ x

- ❖ ค้นปมที่เก็บ x
- ❖ หมุนปมที่เก็บ x ลงไปเป็นใบ
- ❖ ลบใบนั้นทิ้ง



# ต้นไม้ทรีพ : Randomized BST

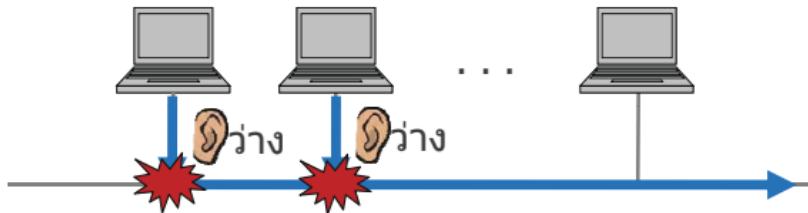
- ❖ เพิ่ม ลบ ค้น ใช้เวลา  $O(\log n)$  - expected time



เพิ่มข้อมูลตามลำดับ 1,2,3,...,500

# Content Resolution

- ❖  $Q_1, Q_2, \dots, Q_n$  ແຍ່ງກັນໃຊ້ R ຮ່ວມກັນ
  - ❖ ທ່ານໃຊ້ R ພຣອມກັນ
  - ❖ ແຕ່ລະ Q ມີວິທີເຂົ້າໃຊ້ R ອຍ່າງໄວ ຖຸກ Q ຄື່ງໄດ້ໃຊ້ R ?
  - ❖ ມາຍເຫດ :  $Q_1, Q_2, \dots, Q_n$  ໄມຄຸຍກັນ



# Content Resolution : Las Vegas



...



```
while(true) {  
    ...  
    access( R );  
    ...  
}
```

...

```
while(true) {  
    ...  
    access( R );  
    ...  
}
```

```
access( R ) {  
    if ( R.available() ) {  
        if ( random(0,1) ≤ p ) {  
            use( R )  
        }  
    }  
}
```

p = ?

## Content Resolution : $p = ?$

- ❖ ให้แต่ละ  $Q$  ต้องการใช้  $R$  ในรอบที่  $t$  ด้วยความน่าจะเป็น  $p$
- ❖ ให้  $S(i, t)$  คือเหตุการณ์ที่
  - ❖  $Q_i$  ต้องการใช้  $R$  ในรอบที่  $t$  และได้ใช้ในรอบที่  $t$

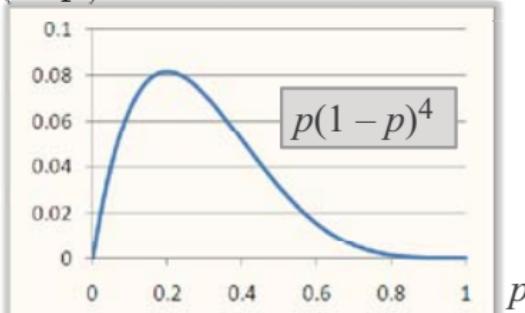
$$\Pr[S(i, t)] = p(1-p)^{n-1}$$

$$0 = (1-p)^{n-1} - p(n-1)(1-p)^{n-2}$$

$$(1-p)^{n-1} = p(n-1)(1-p)^{n-2}$$

$$(1-p) = p(n-1)$$

$$p = \frac{1}{n}$$

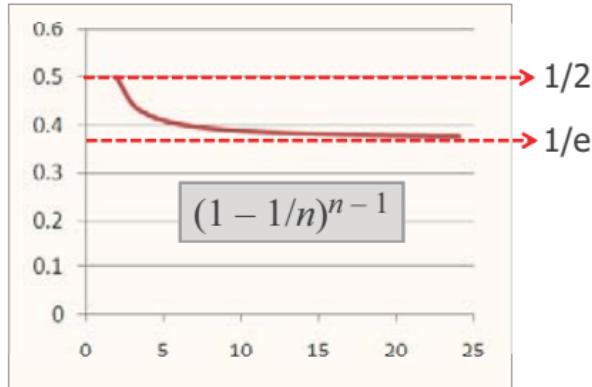


# โอกาสที่ได้ใช้ R

- ❖ ให้แต่ละ Q ต้องการใช้ R ในรอบที่ t  
ด้วยความน่าจะเป็น  $p = 1/n$
- ❖ ให้  $S(i, t)$  คือเหตุการณ์ที่
  - ❖  $Q_i$  ต้องการใช้ R ในรอบที่ t และได้ใช้ในรอบที่ t

$$\begin{aligned} \Pr[S(i, t)] &= p(1 - p)^{n-1} \\ &= \frac{1}{n} \left(1 - \frac{1}{n}\right)^{n-1} \end{aligned}$$

$$\frac{1}{en} \leq \Pr[S(i, t)] \leq \frac{1}{2n}$$



# โอกาสที่ไม่ได้ใช้ R

❖ ให้  $F(i, t)$  คือเหตุการณ์ที่

❖  $Q_i$  ต้องการใช้ R ในรอบที่ 1 ถึง  $t$ , แต่ไม่ได้ใช้เลย

$$\Pr[F(i, t)] = (1 - \Pr[S(i, t)])^t$$

$$(1 - (en)^{-1})^{en} \leq \left(1 - \frac{1}{en}\right)^t \quad \frac{1}{en} \leq \Pr[S(i, t)] \leq \frac{1}{2n}$$

$$(1 - x^{-1})^x \leq e^{-1} \quad \text{เมื่อ } t = en$$

$$\leq e^{-c \ln n} \quad \text{เมื่อ } t = (en)(c \ln n)$$

$$= n^{-c}$$

# จำนวนรอบที่มั่นใจว่าได้ใช้ R

ให้  $F(t)$  คือเหตุการณ์ที่มีอย่างน้อย 1 งานที่ไม่ได้ใช้ R ใน t รอบ

$$\Pr[F(i,t)] \leq n^{-c} \quad \text{เมื่อ } t = (en)(c \ln n)$$

$$\Pr[F(t)] = \sum_{i=1}^n \Pr[F(i,t)]$$

$$\leq n \left(1 - \frac{1}{en}\right)^t$$

$$\leq n \cdot n^{-2} \quad \text{เมื่อ } t = (en)(2 \ln n)$$

$$= \frac{1}{n}$$

สรุป : ถ้าแต่ละงานเข้าใช้ R ด้วยความน่าจะเป็น  $1/n$

เมื่อผ่านไป  $\lceil 2en \ln n \rceil$  รอบ ทุกงานได้ใช้ R อย่างน้อย 1 ครั้ง  
ด้วยความน่าจะเป็นอย่างน้อย  $1 - n^{-1}$

# Randomized N-Queen

```
LV-NQueen( col[1..n] ) {
    for ( i = 1; i <= n; i++) col[i] = i;
    while ( Randomized-NQueen(col) == false ) {}
    print( col )
}

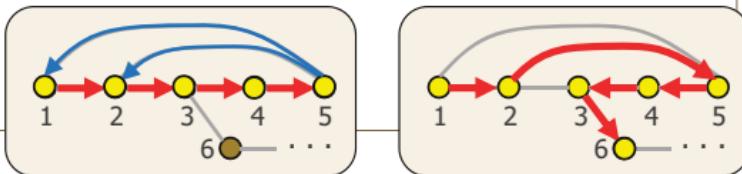
Randomized-NQueen( col[1..n] ) {
    shuffle( col )
    for(k = 2; k <= n; k++) {
        j = k + 1
        while (!isValid(col, k)) {
            if (j > n) return false
            col[k] ↔ col[j];
            j++
        }
    }
    return true
}
```

# Hamiltonian Path : Randomized

```

Randomized-HamiltonianPath( G=(V,E) ) {
    u = ปั้นหนึ่ง (เลือกอย่างสุ่ม) ของ V
    path = < u >
    while ( |path| < |V| ) {
        v = ปั้นปลายของ path
        if (มีปั้นที่ติดกับ v ที่ไม่อยู่ใน path ) {
            u = ปั้นที่ติดกับ v ที่ไม่อยู่ใน path (เลือกแบบสุ่ม)
            path.append( u )
        } else if (มีปั้นที่ติดกับ v) {
            u = ปั้นที่ติดกับ v (เลือกแบบสุ่ม)
            rotatePath( path, u )
        } else {
            return FAIL
        }
    }
}

```



# Hamiltonian Path : Las Vegas

```
LV-HamiltonianPath( G=(V,E) ) {  
    do {  
        path = Randomized-HamiltonianPath( G )  
    } while( path == FAIL )  
    return path  
}
```

# การจับคู่สตริง

สมชาย ประสิทธิ์จุตระกุล

มหาวิทยาลัยเทคโนโลยีราชมงคลรัตนโกสินทร์

# หัวข้อ

นิยามปัญหา

**Brute-force (naïve)**

**Rabin-Karp**

**Finite-automaton**

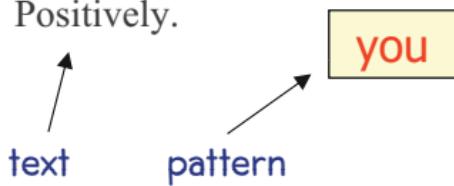
**Knuth-Morris-Pratt**

**Boyer-Moore**

**Horspool**

# Exact String Matching

You are the fairest of your sex,  
Let me be your hero;  
I love you as one over  $x$ ,  
As  $x$  approaches zero.  
Positively.



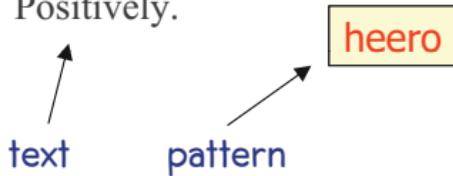
Input

You are the fairest of **your** sex,  
Let me be **your** hero;  
I love **you** as one over  $x$ ,  
As  $x$  approaches zero.  
Positively.

Output

# Approximate String Matching

You are the fairest of your sex,  
Let me be your hero;  
I love you as one over  $x$ .  
As  $x$  approaches zero.  
Positively.



Input

You are the fairest of your sex,  
Let me be your **hero**;  
I love you as one over  $x$ ,  
As  $x$  approaches **zero**.  
Positively.

Output

# ปัญหา String Matching

## ❖ Finite alphabet : $\Sigma$

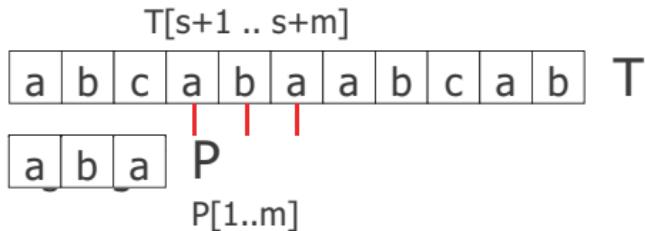
- ❖  $\Sigma = \{a, b, c, \dots, z\}$
- ❖  $\Sigma = \{0, 1\}$
- ❖  $\Sigma = \{A, C, G, T\}, \dots$

ต้องการหา  
all valid shifts  
first valid shift  
last valid shift

## ❖ Text : $T[1..n]$

$$0 \leq s \leq n - m$$

## ❖ Pattern : $P[1..m]$

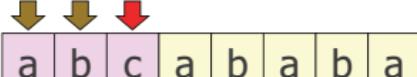
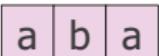


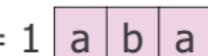
$s$  เป็น valid shift เมื่อ  $P[1..m] = T[s+1 .. s+m]$

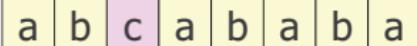
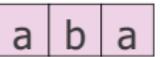
# อัลกอริทึมการเทียบสตริงแบบแม่นตรง

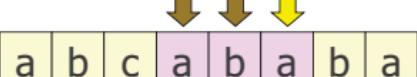
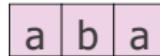
- ✓❖ Brute Force algorithm
- ✓❖ Deterministic Finite Automaton algorithm
- ✓❖ Rabin-Karp algorithm
  - ❖ Shift Or algorithm
  - ❖ Morris-Pratt algorithm
- ✓❖ Knuth-Morris-Pratt algorithm
  - ❖ Simon algorithm
  - ❖ Colussi algorithm
  - ❖ Galil-Giancarlo algorithm
  - ❖ Apostolico-Crochemore algorithm
- ✓❖ Boyer-Moore algorithm
  - ❖ Turbo BM algorithm
  - ❖ Apostolico-Giancarlo algorithm
  - ❖ Reverse Colussi algorithm
- ✓❖ Horspool algorithm
  - ❖ Quick Search algorithm
  - ❖ Tuned Boyer-Moore algorithm
- ❖ Zhu-Takaoka algorithm
- ❖ Berry-Ravindran algorithm
- ❖ Smith algorithm
- ❖ Raita algorithm
- ❖ Reverse Factor algorithm
- ❖ Turbo Reverse Factor algorithm
- ❖ Forward Dawg Matching algorithm
- ❖ Backward Nondeterministic Dawg Matching algorithm
- ❖ Backward Oracle Matching algorithm
- ❖ Galil-Seiferas algorithm
- ❖ Two Way algorithm
- ❖ String Matching on Ordered Alphabets algorithm
- ❖ Optimal Mismatch algorithm
- ❖ Maximal Shift algorithm
- ❖ Skip Search algorithm
- ❖ KMP Skip Search algorithm
- ❖ Alpha Skip Search algorithm
- ❖ ...

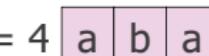
# Brute Force (Naïve) Algorithm

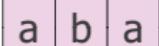
T      
 $s = 0$  

  
T      
 $s = 1$  

  
T      
 $s = 2$  

T      
  $s = 3$  

  
T      
 $s = 4$  

  
T      
  $s = 5$  

# Brute Force (Naïve) Algorithm

```
Naive-String-Matching( T[1..n] , P[1..m] ) {  
    for( s = 0 to n-m ) { ←  $\Theta(n-m)$   
        for( i = 1 to m ) {  
            if (T[s+i] ≠ P[i]) break;  $O(m)$   
        }  
        if (i > m) print( s )  
    }  
}
```

$O(nm)$

worst case :       $T = \text{AAAAAAAAA...AAAAAZ}$   
                         $P = \text{AAAAZ}$

average # of comparisons : (random text and random pattern)

$$d = |\Sigma|$$

$$\frac{1-d^{-m}}{1-d^{-1}}(n-m+1) < 2(n-m+1)$$

# Random T&P : Avg #Cmps

$$\left. \begin{array}{l}
 \text{ตัวที่ 1 ไม่เหมือน} \\
 \text{ตัวแรกเหมือน} \\
 \text{ตัวที่ 2 ไม่เหมือน} \\
 \text{สองตัวแรกเหมือน} \\
 \text{ตัวที่ 3 ไม่เหมือน} \\
 \dots
 \end{array} \right\} \times 1$$

$$\left. \begin{array}{l}
 \left( \frac{1}{d} \right)^1 \left( \frac{d-1}{d} \right) \\
 \left( \frac{1}{d} \right)^2 \left( \frac{d-1}{d} \right) \\
 \left( \frac{1}{d} \right)^3 \left( \frac{d-1}{d} \right) \\
 \dots
 \end{array} \right\} \times 2$$

$$\left. \begin{array}{l}
 \left( \frac{1}{d} \right)^2 \left( \frac{d-1}{d} \right) \\
 \left( \frac{1}{d} \right)^3 \left( \frac{d-1}{d} \right) \\
 \dots
 \end{array} \right\} \times 3$$

$$\sum_{k=1}^m \frac{k}{d^{k-1}} \left( \frac{d-1}{d} \right)$$

$$\left. \begin{array}{l}
 m-1 \text{ ตัวแรกเหมือน} \\
 \text{ตัวที่ } m \text{ ไม่เหมือน}
 \end{array} \right\} + \left( \frac{1}{d} \right)^{m-1} \left( \frac{d-1}{d} \right) \times m$$

$$\left. \begin{array}{l}
 \text{เหมือนกันทั้ง } m \text{ ตัว} \\
 \dots
 \end{array} \right\} \times m = \frac{m}{d^m}$$

## Random T&P : Avg #Cmps

$$\frac{m}{d^m} + \sum_{k=1}^m \frac{k}{d^{k-1}} \left( \frac{d-1}{d} \right) = \frac{m}{d^m} + \left( 1 - \frac{1}{d} \right) \sum_{k=1}^m \frac{k}{d^{k-1}} = ?$$

$$\sum_{k=0}^m x^k = \frac{1-x^{m+1}}{1-x}$$

$$\sum_{k=1}^m kx^{k-1} = \frac{-(1-x)(m+1)x^m + (1-x^{m+1})}{(1-x)^2}$$

$$= \frac{1 - (m+1)x^m + mx^{m+1}}{(1-x)^2}$$

$$= \frac{1 - x^m + mx^m(x-1)}{(1-x)^2}$$

# Random T&P : Avg #Cmps

$$\frac{m}{d^m} + \left(1 - \frac{1}{d}\right) \sum_{k=1}^m \frac{k}{d^{k-1}} = \frac{1 - d^{-m}}{1 - d^{-1}}$$

$$\sum_{k=1}^m kx^{k-1} = \frac{1 - x^m + mx^m(x-1)}{(1-x)^2}$$

$$\begin{aligned}
 mx^m + (1-x) \sum_{k=1}^m kx^{k-1} &= mx^m + (1-x) \left( \frac{1 - x^m + mx^m(x-1)}{(1-x)^2} \right) \\
 &= mx^m + \left( \frac{1 - x^m + mx^m(x-1)}{(1-x)} \right) \\
 &= \frac{1 - x^m}{1 - x} \quad \longrightarrow \boxed{\text{ให้ } x = d^{-1}}
 \end{aligned}$$

# Random T&P : Avg #Cmps

```
Naive-String-Matching( T[1..n], P[1..m] ) {
    for( s = 0 to n-m ) {
        for( i = 1 to m ) {
            if (T[s+i] ≠ P[i]) break;
        }
        if (i > m) print( s )
    }
}
```

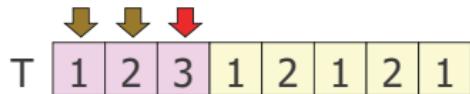
$$\frac{1-d^{-m}}{1-d^{-1}}$$

$$\frac{1-d^{-m}}{1-d^{-1}}(n-m+1) < 2(n-m+1)$$

$$\frac{1-d^{-m}}{1-d^{-1}} < \frac{1}{1-d^{-1}}, \quad d \geq 2$$

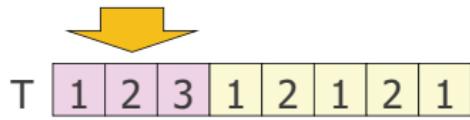
$$\leq \frac{1}{1-2^{-1}} = 2$$

# Rabin-Karp Matcher



1	2	1
---	---	---

เปรียบเทียบ 1 กับ 1,  
2 กับ 2,  
3 กับ 1



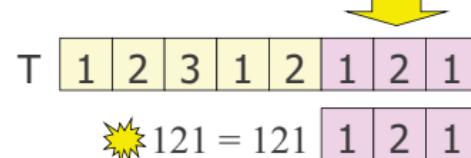
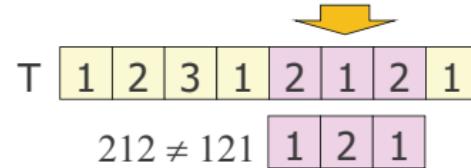
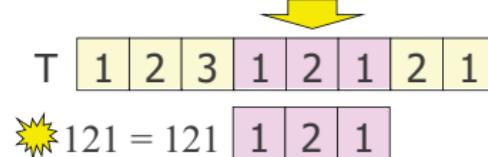
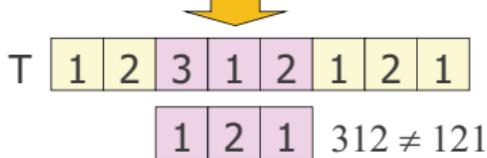
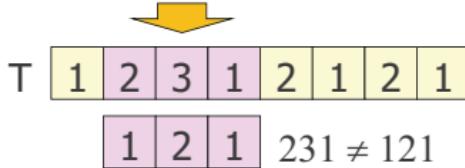
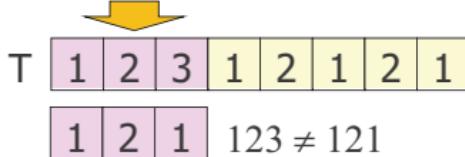
123

121

1	2	1
---	---	---

เปรียบเทียบจำนวน 123 กับ 121

# Rabin-Karp Algorithm



ถ้า pattern เป็นตัวอักษร หรือ  
ถ้าเป็นตัวเลข แต่มีขนาดใหญ่เกิน int เกิน long } จะทำอย่างไร ?

# Hashing

T [a | b | c | a | b | a | b | a]

$h("abc") \rightarrow 348 \neq 78$

T [a | b | c | a | b | a | b | a]

$h("bca") \rightarrow 519 \neq 78$

T [a | b | c | a | b | a | b | a]

$h("cab") \rightarrow 13 \neq 78$

T [a | b | c | a | b | a | b | a]

$h("aba") \rightarrow 78 = 78$

P [a | b | a]

$h("aba")$

78

T [a | b | c | a | b | a | b | a]

$h("bab")$

78 = 78

spurious hit

ตั้งนั้น ถ้า hit ต้อง  
เทียบแต่ละ char ด้วย

อีกปัญหา : การคำนวณ  $h(p)$   
ที่ใช้เวลาตามขนาดของ  $p$

## การคำนวณค่า $h$ : incremental

- ❖  $h(s) = (\text{มอง } s \text{ เป็นจำนวน ในระบบเลขฐาน})$
- ❖  $h("abc") = (1 \times 37^2 + 2 \times 37^1 + 3 \times 37^0) = 1446$
- ❖  $h("bce") = (2 \times 37^2 + 3 \times 37^1 + 5 \times 37^0)$   
 $= 37 \times (\cancel{1 \times 37^2} + 2 \times 37^1 + 3 \times 37^0) + 5 \times 37^0$   
 $= 37 \times (h("abc") - 1 \times 37^2) + 5 \times 37^0$

เลื่อนตำแหน่ง "bc"  
ไปทางซ้าย

ลบ "a"

เพิ่ม "e"



$$h("...e") = 37 \times (h("a...") - 1 \times 37^{m-1}) + 5 \times 37^0$$



# Rabin-Karp Algorithm

```

Rabin-Karp-Matching( T[1..n] , P[1..m] , d ) {
    ht = 0; hp = 0
    for ( i = 1 to m ) {
        ht = d*ht + T[i]
        hp = d*hp + P[i]
    }
    for ( s = 0 to n - m ) {
        if (ht == hp) {
            for ( i = 1 to m )
                if (T[s+i] ≠ P[i]) break
            if (i > m) print( s )
        }
        if (s < n-m) {
            ht = d*(ht - T[s+1]*dm-1) + T[s+m+1]
        }
    }
}

```

$d = |\Sigma|$

$h(\dots e) = 37 \times ( h(a\dots) - 1 \times 37^{m-1} ) + 5 \times 37^0$

ถ้า  $m$  มีค่ามาก ขนาด  
ของ  $ht$  และ  $hp$  ก็ใหญ่

# Rabin-Karp Algorithm

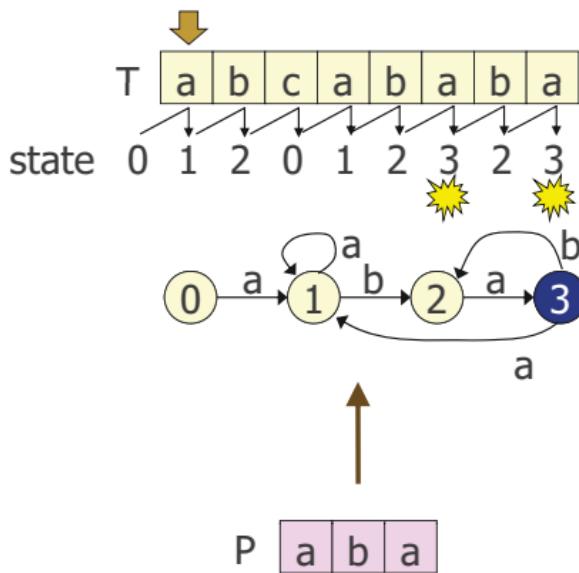
```
Rabin-Karp-Matching( T[1..n] , P[1..m] , d, q ) {
    ht = 0; hp = 0; dm1 = dm-1 % q
    for ( i = 1 to m ) {
        ht = (d*ht + T[i]) % q
        hp = (d*hp + P[i]) % q
    }
    for ( s = 0 to n - m ) {
        if (ht == hp) {
            for ( i = 1 to m )
                if (T[s+i] ≠ P[i]) break
            if (i > m) print( s )
        }
        if (s < n-m) {
            ht = (d*(ht - T[s+1]*dm1) + T[s+m+1]) % q
        }
    }
}
```

$h(\dots e) = 37 \times (h(a\dots) - 1 \times 37^{m-1}) + 5 \times 37^0$

worst case  $\Theta(nm)$

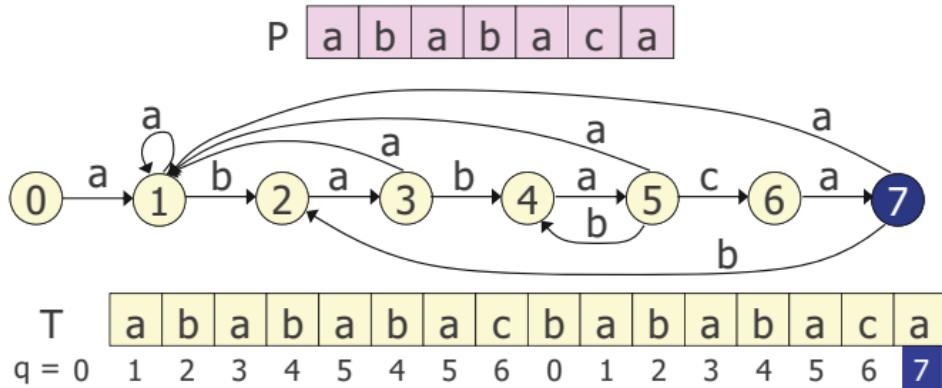
avg case  $O(n + m)$

# Finite Automaton Matcher



สร้าง state transition diagram จาก pattern

# Finite Automaton Matcher



```
Finite-Automaton-Matching(T[1..n], δ, m) {  
    q = 0  
    for( i = 1 to n ) {  
        q = δ(q, T[i])  
        if (q == m) print( i-m )  
    }  
}
```

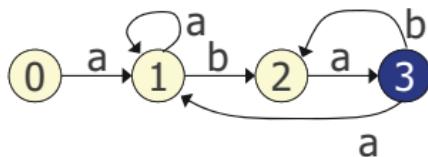
transition function

สร้าง transition function อย่างไร ?

$\Theta(n)$

# Transition Function

P [a | b | a]



$$\Sigma = \{a, b, c\}$$

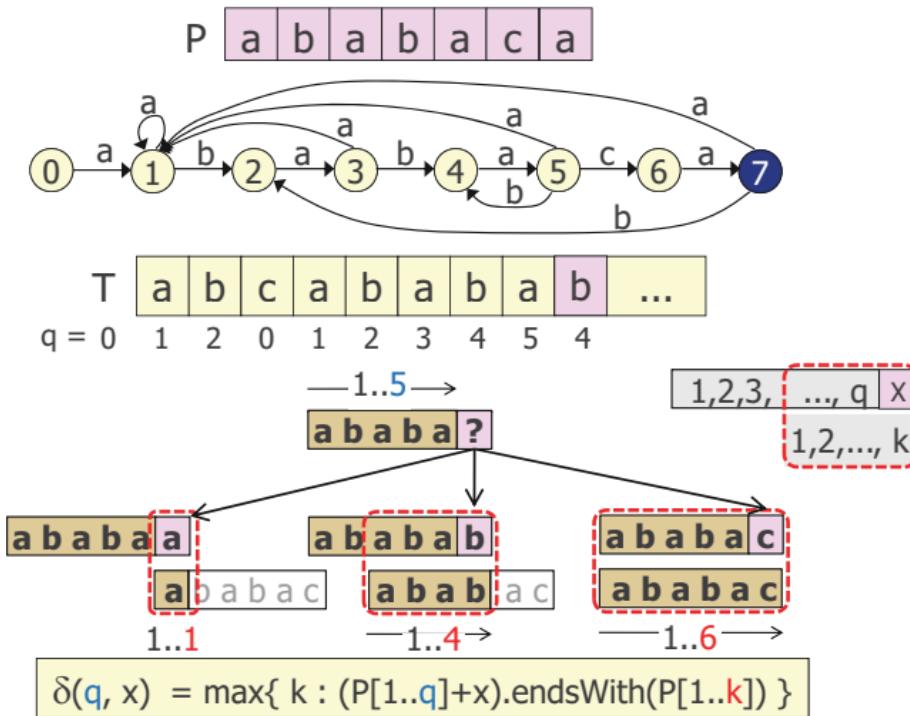
ตารางมีขนาด  
 $(m+1)|\Sigma|$

input

state	$ \Sigma $		
	a	b	c
0	1	0	0
1	1	2	0
2	3	0	0
3	1	2	0

$\delta(state, input)$

# Transition Function



# ตัวอย่าง Transition Function

$$\delta(q, x) = \max\{ k : (P[1..q]+x).endsWith(P[1..k]) \}$$

state a b c

0	1	0	0
1	1	2	0
2	3	0	0
3	1	4	0
4	5	0	0
5	1	4	6
6	7	0	0
7	1	2	0

P a b a b a c a

$$\delta(5, "b") = 4$$

$$P[1..5] + "b" = ababa$$

~~a~~

~~ab~~

~~aba~~

~~abab~~

~~ababa~~

~~ababac~~

~~ababaca~~

P[1..4]

สร้างแบบตรงไปตรงมาใช้เวลา  $O(m^3|\Sigma|)$

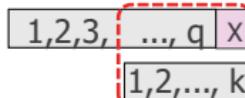
มีวิธีสร้างที่ใช้เวลา  $O(m|\Sigma|)$

รวมการค้นด้วย ใช้เวลา  $O(n + m|\Sigma|)$

# Knuth-Morris-Pratt Matcher

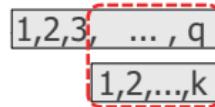
## ❖ ของเดิม (transition function)

- ❖  $\delta(q, x) = \max\{ k : (P[1..q]+x).endsWith(P[1..k]) \}$
- ❖ ตารางมีขนาด  $(m+1)|\Sigma|$
- ❖ ใช้เวลาสร้าง  $O(m|\Sigma|)$



## ❖ ของใหม่ (prefix function)

- ❖  $\pi(q) = \max\{ k : k < q \text{ and } (P[1..q]).endsWith(P[1..k]) \}$
- ❖ match มาได้ q ตัว แต่ mismatch ตัวที่ q+1 ให้ทำต่อที่  $P[\pi(q)+1]$
- ❖ ตารางมีขนาด m
- ❖ ใช้เวลาสร้าง  $\Theta(m)$



## ❖ สร้าง $\pi$ และค้น ใช้เวลารวม $\Theta(n+m)$

# การใช้ Prefix Function

	P	a b a b a c a	$\pi$	0 0 1 2 3 0 1	1	2	3	4	5	6	7
	T	a b a b a a b a b a b a c a a b a b a c a									

T a b a b a a b a b a b a c a  
a b a b a c a  $\pi[5] = 3$

T a b a b a a b a b a b a c a  
a b a b a c a  $\pi[3] = 1$

T a b a b a a b a b a b a c a  
a b a b a c a  $\pi[1] = 0$

T a b a b a a b a b a b a c a  
a b a b a c a  $\pi[5] = 3$

T a b a b a a b a b a b a c a  
a b a b a c a

# KMP-Matcher

```
KMP-Matcher( T[1..n] , P[1..m] ) {
    π = Compute-Prefix-Function(P) // Θ(m)
    q = 0
    for ( i = 1 to n ) {
        while (q > 0 AND P[q+1] ≠ T[i]) {
            q = π[q]
        }
        if (P[q+1] == T[i]) q++
        if (q == m) {
            print( i - m )
            q = π[q]
        }
    }
}
```

 $\Theta(n + m)$ 

$$\pi[4] = 2 \longleftrightarrow$$

## การออคแบบอัลกอริทึม

$$\pi(q) = \max\{ k : k < q \text{ and } (P[1..q]).endsWith(P[1..k]) \}$$

$q = 1$

a	b	a	b	a	c	a
a	b	a	b	a	c	a

$\pi[1] = 0$

$q = 2$

a	b	a	b	a	c	a
a	b	a	b	a	c	a

$\pi[2] = 0$

$q = 3$

a	b	a	b	a	c	a
a	b	a	b	a	c	a

$\pi[3] = 1$

$q = 4$

a	b	a	b	a	c	a
a	b	a	b	a	c	a

$\pi[4] = 2$

$q = 5$

a	b	a	b	a	c	a
a	b	a	b	a	c	a

$\pi[5] = 3$

$q = 6$

a	b	a	b	a	c	a
a	b	a	b	a	c	a

$\pi[6] = 0$

$q = 7$

a	b	a	b	a	c	a
a	b	a	b	a	c	a

$\pi[7] = 1$

$P[1..q]$

$P[1..k], k < q$

$$\pi(q) = \max\{ k : k < q \text{ and } (P[1..q]).endsWith(P[1..k]) \}$$

a	t	c	a	c	a	t	c	a	t	c	a
---	---	---	---	---	---	---	---	---	---	---	---

$$\pi[1] = 0$$

a	t	c	a	c	a	t	c	a	t	c	a
a	t	c	a	c	a	t	c	a	t	c	a

$$\pi[2] = 0$$

a	t	c	a	c	a	t	c	a	t	c	a
a	t	c	a	c	a	t	c	a	t	c	a

$$\pi[3] = 0$$

```
Compute-Prefix-Function( P[1..m] ) {  
     $\pi[1] = 0$ ;  $k = 0$   
    for ( q = 2 to m ) {  
  
        ...  
  
        if ( P[k+1] == P[q] )  $k++$   
         $\pi[q] = k$   
    }  
    return  $\pi$ 
```

$$\pi(q) = \max\{ k : k < q \text{ and } (P[1..q]).\text{endsWith}(P[1..k]) \}$$

a	t	c	a	c	a	t	c	a	t	c	a
---	---	---	---	---	---	---	---	---	---	---	---

$$\pi[4] = 1$$

	a	t	c	a	c	a	t	c	a	t	c	a
--	---	---	---	---	---	---	---	---	---	---	---	---

a	t	c	a	c	a	t	c	a	t	c	a
---	---	---	---	---	---	---	---	---	---	---	---

$$\pi[5] = 0$$

	a	a	t	c	a	c	a	t	c	a	t	c	a
--	---	---	---	---	---	---	---	---	---	---	---	---	---

a	t	c	a	c	a	c	a	t	c	a	t	c	a
---	---	---	---	---	---	---	---	---	---	---	---	---	---

$$\pi[6] = 1$$

	a	t	c	a	c	a	t	c	a	t	c	a
--	---	---	---	---	---	---	---	---	---	---	---	---

```
Compute-Prefix-Function( P[1..m] ) {
```

```
     $\pi[1] = 0$ ;  $k = 0$ 
```

```
    for ( q = 2 to m ) {
```

```
        while ( k > 0 and P[k+1]  $\neq$  P[q] ) {
```

```
            k =  $\pi[k]$ 
```

```
        }
```

```
        if ( P[k+1] == P[q] ) k++
```

```
         $\pi[q] = k$ 
```

```
    }
```

```
    return  $\pi$ 
```

$$\pi(q) = \max\{ k : k < q \text{ and } (P[1..q]).\text{endsWith}(P[1..k]) \}$$

a	t	c	a	c	a	t	c	a	t	c	a
---	---	---	---	---	---	---	---	---	---	---	---

$$\pi[7] = 2$$

a	t	c	a	c	a	t	c	a	t	c	a
---	---	---	---	---	---	---	---	---	---	---	---

a	t	c	a	c	a	t	c	a	t	c	a
---	---	---	---	---	---	---	---	---	---	---	---

$$\pi[8] = 3$$

a	t	c	a	c	a	t	c	a	t	c	a
---	---	---	---	---	---	---	---	---	---	---	---

a	t	c	a	c	a	t	c	a	t	c	a
---	---	---	---	---	---	---	---	---	---	---	---

$$\pi[9] = 4$$

a	t	c	a	c	a	t	c	a	t	c	a
---	---	---	---	---	---	---	---	---	---	---	---

```
Compute-Prefix-Function( P[1..m] ) {
```

```
     $\pi[1] = 0$ ;  $k = 0$ 
```

```
    for (  $q = 2$  to  $m$  ) {
```

```
        while (  $k > 0$  and  $P[k+1] \neq P[q]$  ) {
```

```
             $k = \pi[k]$ 
```

```
}
```

```
        if (  $P[k+1] == P[q]$  )  $k++$ 
```

```
         $\pi[q] = k$ 
```

```
}
```

```
return  $\pi$ 
```

$$\pi(q) = \max\{ k : k < q \text{ and } (P[1..q]).\text{endsWith}(P[1..k]) \}$$

a	t	c	a	c	a	t	c	a	t	c	a
						c	a	t	c	a	t

$\pi[9] = 4 \rightarrow P[1..4] = P[6..9] \rightarrow \text{ทดสอบ } P[4+1] = P[10] ?$

$\pi[4] = 1 \rightarrow P[1..1] = P[4..4] \rightarrow \text{ทดสอบ } P[1+1] = P[10] ?$

a	t	c	a	c	a	t	c	a	t	c	a
						a	t	c	a	c	a

$\pi[10] = 2$

```
Compute-Prefix-Function( P[1..m] ) {
     $\pi[1] = 0; k = 0$ 
    for ( q = 2 to m ) {
        while ( k > 0 and P[k+1]  $\neq$  P[q] ) {
            k =  $\pi[k]$ 
        }
        if ( P[k+1] == P[q] ) k++
         $\pi[q] = k$ 
    }
    return  $\pi$ 
}
```

$$\pi(q) = \max\{ k : k < q \text{ and } (P[1..q]).\text{endsWith}(P[1..k]) \}$$

a t c a c a t c a t c a

$$\pi[11] = 3$$

a t c a c a t c a t c a

a t c a c a t c a t c a

$$\pi[12] = 4$$

a t c a c a t c a t c a

```
Compute-Prefix-Function( P[1..m] ) {
     $\pi[1] = 0; k = 0$ 
    for ( q = 2 to m ) {
        while ( k > 0 and P[k+1] != P[q] ) {
            k =  $\pi[k]$ 
        }
        if ( P[k+1] == P[q] ) k++
         $\pi[q] = k$ 
    }
    return  $\pi$ 
}
```

$\Theta(m)$

# Boyer-Moore Matcher

## Knuth-Morris-Pratt (left-to-right)

A STRING SEARCHING EXAMPLE CONSISTING OF ...  
STING

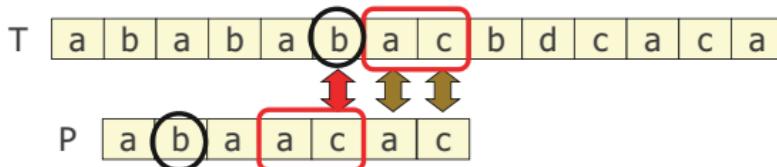


## Boyer Moore (right-to-left)

A STRING SEARCHING EXAMPLE CONSISTING OF ...  
STING

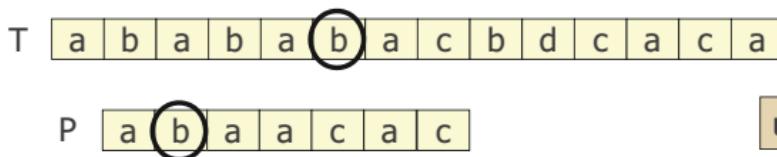


# Boyer-Moore : Two heuristics



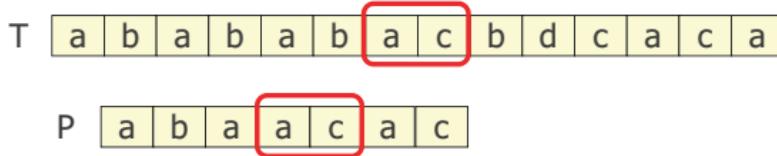
**Bad character**

bad-character แนะนำให้เลื่อน 3 ตำแหน่ง



**Good suffix**

good-suffix แนะนำให้เลื่อน 2 ตำแหน่ง



# Bad-Character Heuristics

T [ a b a b a b ] a c b d c a c a a c a

P [ a b a a c a c ]

1 2 3 4 5 6 7

bad-character แนะนำให้เลื่อน

$$5 - 2 = 3 \text{ ตำแหน่ง}$$

T [ a b a b a b ] a c b d c a c a a c a

P [ a b a a c a c ]

$$\lambda['b'] = 2$$

$$\lambda[x] = P[1..m].lastIndexOf(x)$$

P [ a b a a c a c ]

1

2

3

4

5

6

7

$\Sigma$	a	b	c	d
$\lambda$	6	2	7	0

# Bad-Character Heuristics

T	a b a b a b a c b d c a c a a c a	P	a b a a c a c		bad-character แนะนำให้เลื่อน $5 - 2 = 3$ ตำแหน่ง
		1 2 3 4 5 6 7			
T	a b a b a b a b a c b d c a c a a c a	เลื่อน 3 ข้อง P	a b a a c a c		bad-character แนะนำให้เลื่อน $6 - 0 = 6$
		1 2 3 4 5 6 7			
T	a b a b a b a b a c b d c a c a a c c	เลื่อน 6 ข้อง P	a b a a c a c		bad-character แนะนำให้เลื่อน $3 - 7 = -4$ (ignored)
$\Sigma$	a b c d				
$\lambda$	6 2 7 0				

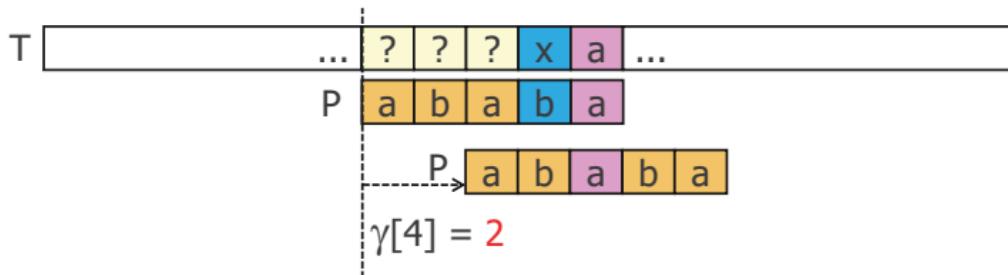
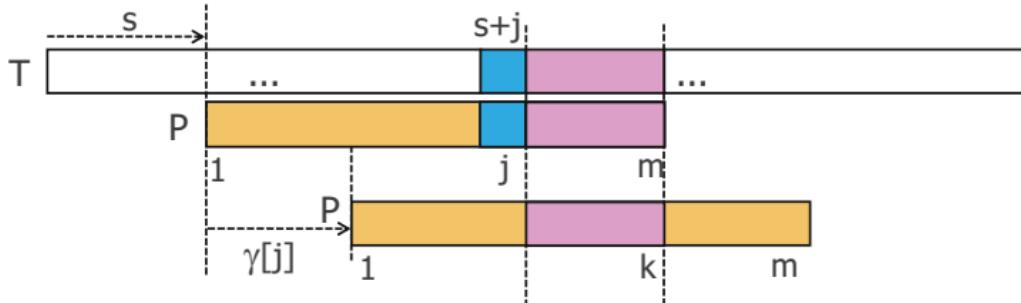
# Good-Suffix Heuristics

T .. . . . . b a c .. . . . .  
 P a a c a d a c  
 1 2 3 4 5 6 7

$\gamma[5] = 4$   
 mismatch ตำแหน่งที่ 5

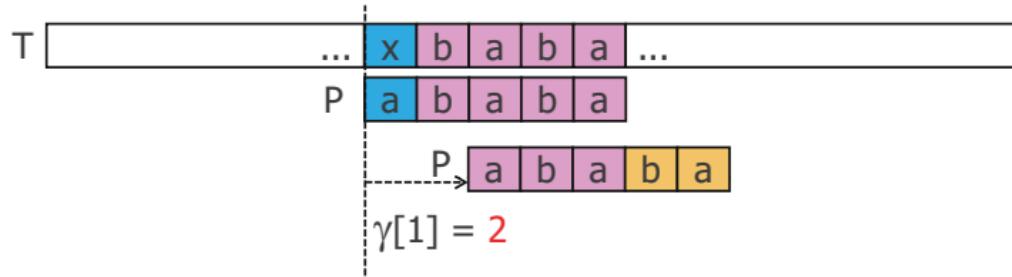
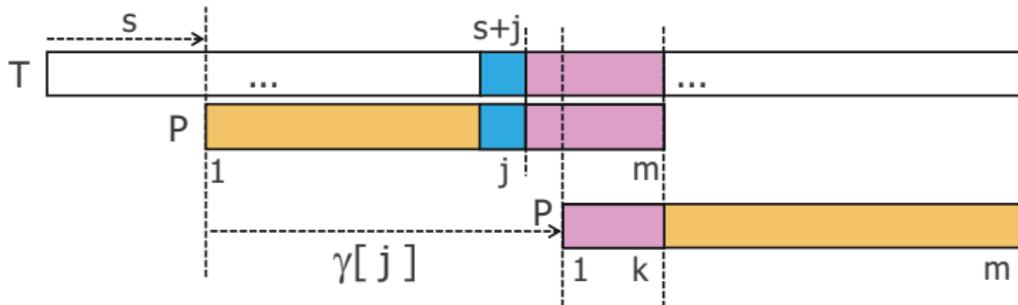
mismatch ตำแหน่งที่ 5  
good-suffix แนะนำให้เลื่อน  
4 ตำแหน่ง

# Good-Suffix Heuristics



$$\gamma[j] = m - \max \{ k : 0 \leq k < m \text{ and } P[1..k].\text{endsWith}(P[j+1..m]) \}$$

# Good-Suffix Heuristics

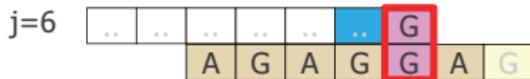


$$\gamma[j] = m - \max \{ k : 0 \leq k < m \text{ and } P[j+1..m].\text{endsWith}(P[1..k]) \}$$

$$\gamma[j] = m - \max \{ k : 0 \leq k < m \text{ and } P[j+1..m] \sim P[1..k] \}$$

P A G A G G G A G

$$\gamma[7] = 1$$



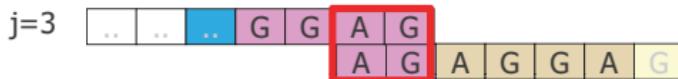
$$\gamma[6] = 7 - 5 = 2$$



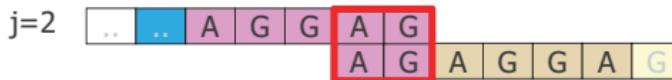
$$\gamma[5] = 7 - 4 = 3$$



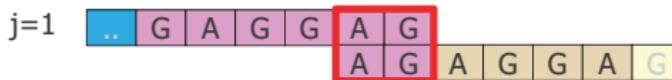
$$\gamma[4] = 7 - 4 = 3$$



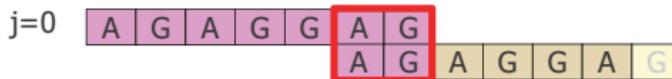
$$\gamma[3] = 7 - 2 = 5$$



$$\gamma[2] = 7 - 2 = 5$$



$$\gamma[1] = 7 - 2 = 5$$



$$\gamma[0] = 7 - 2 = 5$$

# Boyer-Moore Matcher

```

Boyer-Moore-Matcher(T[1..n], P[1..m], Σ)
λ = LAST-OCCURRENCE(P, m, Σ)    // O(|Σ|+m)
γ = GOOD-SUFFIX(P, m)           // O(m)
s = 0
while( s ≤ n - m ) {
    j = m
    while( j > 0 and P[j] = T[s+j]) j--
    if (j==0) {
        print( s )
        s = s + γ[0]
    } else {
        s = s + max(γ[j], j - λ[T[s+j]])
    }
}

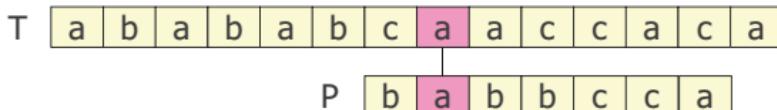
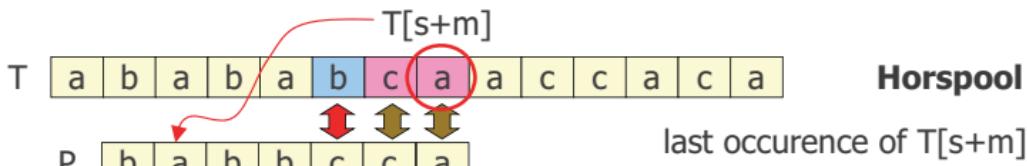
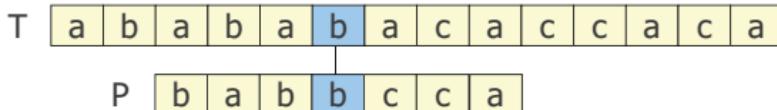
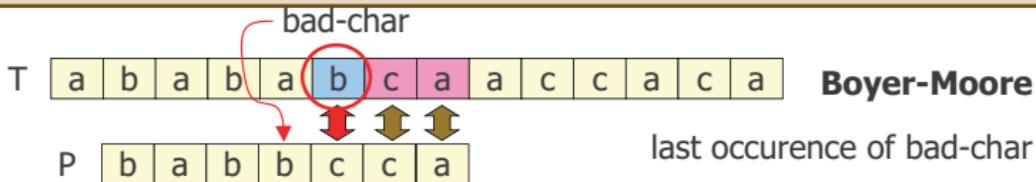
```

worst :  $O((n - m + 1)m + |\Sigma|)$ best :  $O(n/m + m + |\Sigma|)$

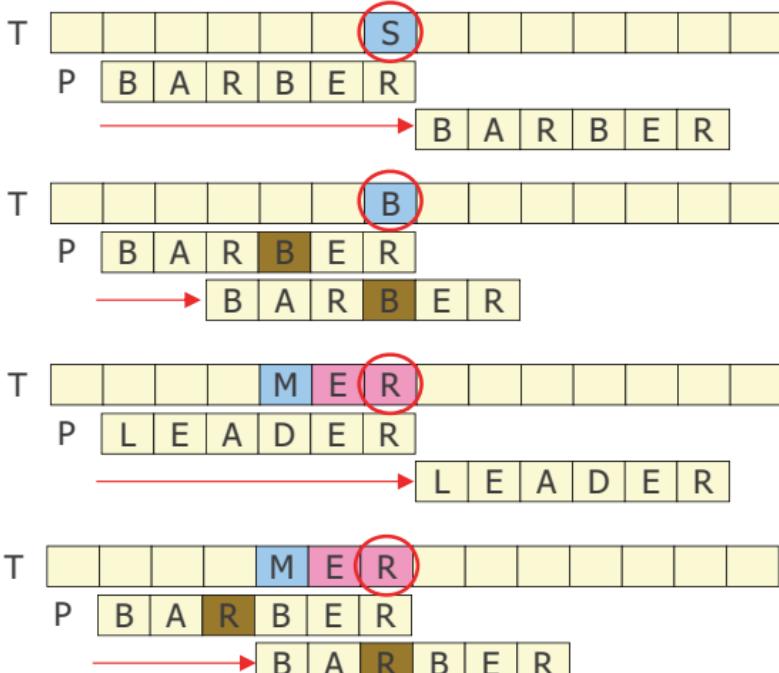
# Boyer-Moore Matcher

```
Boyer-Moore-Matcher(T[1..n], P[1..m], Σ)
λ = LAST OCCURRENCE(P, m, Σ)
γ = GOOD SUFFIX(P, m)
s = 0
while( s ≤ n - m ) {
    j = m
    while( j > 0 and P[j] = T[s+j] ) j--
    if (j==0) {
        print( s )
        s = s + γ[0]; 1
    } else {
        s = s + max(γ[j], j - λ[T[s+j]]); 1
    }
}
```

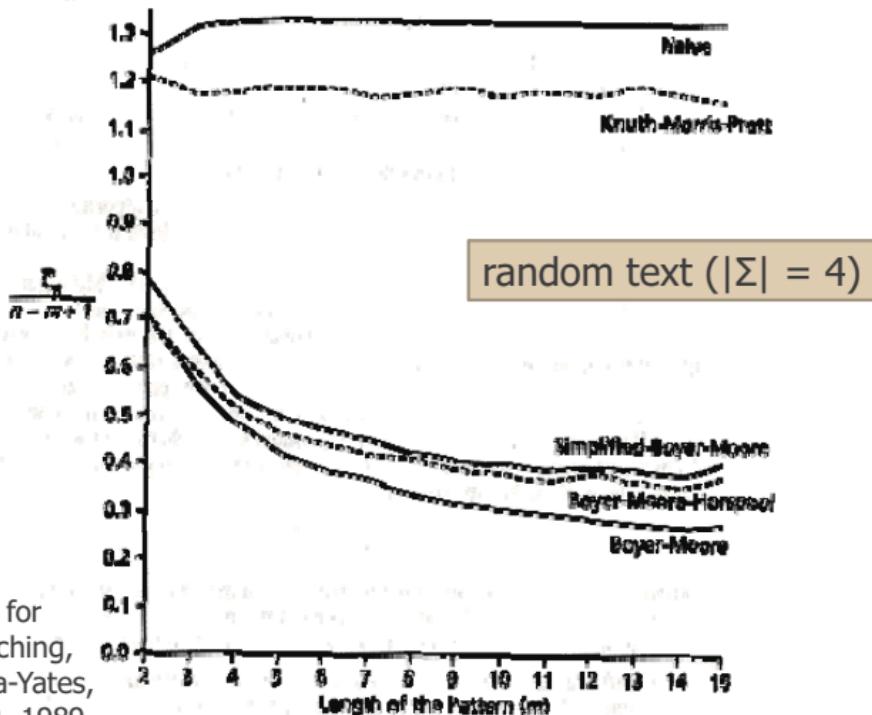
# Horspool Matcher



# Horspool Matcher

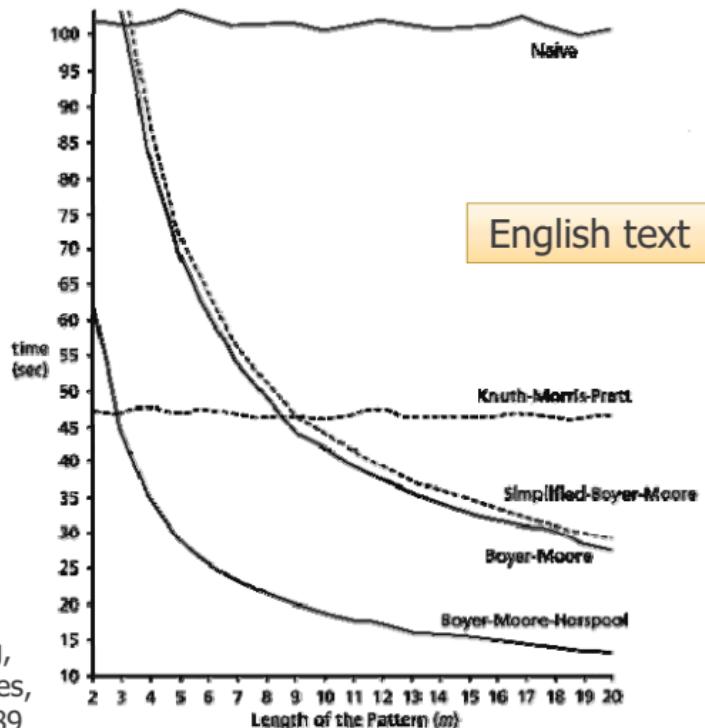


# ประสิทธิภาพการทำงาน



Algorithms for  
string searching,  
R. A. Baeza-Yates,  
ACM SIGIR, 1989

# ประสิทธิภาพการทำงาน



Algorithms for  
string searching,  
R. A. Baeza-Yates,  
ACM SIGIR, 1989